An extended abstract of this paper is published in the proceedings of the 40th International Colloquium on Automata, Languages and Programming—ICALP 2013. This is the full version.

Outsourced Pattern Matching

Sebastian Faust* Carmit Hazay† Daniele Venturi‡

June 18, 2014

Abstract

In secure delegatable computation, computationally weak devices (or clients) wish to outsource their computation and data to an *untrusted* server in the cloud. While most earlier work considers the general question of how to securely outsource *any* computation to the cloud server, we focus on *concrete* and *important* functionalities and give the first protocol for the *pattern matching* problem in the cloud. Loosely speaking, this problem considers a text T that is outsourced to the cloud S by a sender SEN. In a query phase, receivers $\mathrm{REC}_1, \ldots, \mathrm{REC}_l$ run an efficient protocol with the server S and the sender SEN in order to learn the positions at which a pattern of length m matches the text (and nothing beyond that). This is called the *outsourced pattern matching* problem which is highly motivated in the context of delegatable computing since it offers storage alternatives for massive databases that contain confidential data (e.g., health related data about patient history).

Our constructions are *simulation-based secure* in the presence of semi-honest and malicious adversaries (in the random oracle model) and limit the communication in the query phase to $O(m)$ bits plus the number of occurrences—which is optimal. In contrast to generic solutions for delegatable computation, our schemes do not rely on fully homomorphic encryption but instead use novel ideas for solving pattern matching, based on a reduction to the subset sum problem. Interestingly, we do not rely on the hardness of the problem, but rather we exploit instances that are solvable in polynomial-time. A follow-up result demonstrates that the random oracle is essential in order to meet our communication bound.

*EPFL, Lausanne, Switzerland. Email: sebastian.faust@gmail.com.

†Faculty of Engineering, Bar-Ilan University, Israel. Email: carmit.hazay@biu.ac.il.

‡Computer Science Department, Sapienza University of Rome, Italy. Email: daniele.venturi@uniroma1.it.

Contents

1 Introduction

Background on outsourced secure computation. The problem of securely outsourcing computation to an *untrusted* server gained momentum with the recent penetration of *cloud computing* services. In cloud computing, clients can lease computing services on demand rather than maintaining their own infrastructure. While such an approach naturally has numerous advantages in cost and functionality, it is crucial that the outsourcing mechanism enforces privacy of the outsourced data and integrity of the computation. Solutions based on cryptographic techniques have been put forward with the concept of *secure delagatable computation* [AIK10, CKV10, GGP10, AJLA+12], which lately has received broad attention within the cryptographic research community.

In secure delegatable computation, computationally weak devices (or clients) wish to outsource their computation and data to an *untrusted* server in the cloud. The ultimate goal in this setting is to design efficient protocols that minimize the computational overhead of the clients and instead rely on the extended resources of the server. Of course, the amount of work invested by the client in order to verify the correctness of the computation shall be *substantially* smaller than running the computation by itself. Indeed, if this was not the case then the client could carry out the computation itself without putting confidentiality and integrity of its data at risk. Another ambitious challenge of delegatable computation is to design protocols that minimize the *communication* between the cloud and the client, while using an optimal number of rounds. This becomes of particular importance with the proliferation of Smartphone technology and mobile broadband internet connections, as for mobile devices communication and data connectivity is often the more severe bottleneck.

Most recent works in the area of delegatable computation propose solutions to securely outsource *any functionality* to an untrusted server [AIK10, CKV10, GGP10]. While this of course is the holy grail in delegatable computation, such generic solutions often suffer from rather poor efficiency and high com-

putation overhead due to the use of fully homomorphic encryption [Gen09]. Furthermore, these solution concepts typically examine a restricted scenario where a *single client* outsources its computation to an *external untrusted server*. Another line of works studies an extended setting with *multiple clients* that mutually distrust each other and wish to securely outsource a joint computation on their inputs with reduced costs [KMR11, KMR12, LATV12, AJLA+12, CKKC13]. Of course, also in this more complex setting with multiple clients recent constructions build up on fully homomorphic encryption (or consider a more restricted setting where the clients do not collude with the server, or one of the clients "works harder").

To move towards more practical schemes, we may give up on outsourcing arbitrary computation to the cloud, but instead focus on particularly efficient constructions for specific important functionalities. This approach has the potential to avoid the use of fully homomorphic encryption (FHE) by exploiting the structure of the particular problem we intend to solve. Some recent works have considered this question and proposed schemes for polynomial evaluation and keywords search [BGV11], set operations [PTT11] and linear algebra [Moh11]. While these schemes are more efficient than the generic constructions mentioned above, they typically only achieve very limited privacy or do not support multiple distrusting clients.

In this paper, we follow this line of work and provide the first protocols for *pattern matching* in the cloud. The problem of *outsourced pattern matching* is highly motivated in the context of delegatable computing since it offers storage alternatives for massive databases that contain confidential data (e.g., health related data about patient history). In contrast to most earlier works, our constructions achieve a high level of security, while avoiding the use of FHE and minimizing the amount of communication between the parties. We emphasize that even with the power of FHE it is not clear how to get down to optimal communication complexity in two rounds.[1]

Pattern matching in the cloud. The problem of pattern matching considers a text T of length n and a pattern of length m with the goal to find all the locations where the pattern matches the text. Algorithms for pattern matching have been widely studied for decades due to its broad applicability [KMP77, BM77]. Lately, researchers started to look at this problem in the context of secure two-party computation [TPKC07, HL10, GHS10, KM10, HT10] due to growing interests in private text search. In this setting, one party holds the text whereas the other party holds the pattern and attempts to learn all the locations of the pattern in the text (and only that), while the party holding the text learns nothing about the pattern. Unfortunately, these solutions are not directly applicable in the cloud setting, mostly because the communication overhead per search query grows linearly with the text length. Moreover, the text holder delegates its work to an external untrusted server and cannot control the content of the server's responses.

To be precise, in the outsourced setting we consider a set of clients comprised from a sender SEN and a set of receivers $(\mathrm{REC}_1, \ldots, \mathrm{REC}_l)$ that interact with a server S in the following way. In a *setup phase* the sender SEN uploads a preprocessed text to an external server S. This phase is run only once and may be costly in terms of computation and communication. In a *query phase* the receivers $\mathrm{REC}_1, \ldots, \mathrm{REC}_l$ query the text by searching patterns and learn the matched text locations. The main two goals are as follows:

1. *Simulation-based security:* We model outsourced pattern matching by a strong simulation-based security definition (cf. Section 3) that captures all security concerns. Namely, we define a new reactive outsourced pattern matching functionality $\mathcal{F}_{\mathrm{OPM}}$ that ensures the secrecy and integrity of the outsourced text and patterns. For instance, a semi-honest server does not gain any information about the text and patterns, except of what it can infer from the answers to the search queries (this is formalized more accurately below). If the server is maliciously corrupted the functionality implies the correctness of the queries' replies as well. As in the standard secure computation setting, simulation-based

[1] Namely, a two-rounds solution based on FHE would need a circuit that tolerates the worst case output size, which in pattern matching implies a maximal number of matches that is proportional to the length of the text (or the database).

modeling is simpler and stronger than game-based definitions. Specifically, game-based definitions become more cumbersome since the corruption of different parties implies different security threats, e.g., privacy with respect to a malicious server is not the same notion of privacy we expect to hold when SEN is corrupted. Thus a different security definition is required for each party. On the other hand simulation-based security is harder to achieve, especially in the outsourced setting. As the simulator must commit first to the data stored on the server, and yet, be able to simulate the server's view as in the real interaction. We further elaborate on these difficulties below.

2. *Sublinear communication complexity during query phase:* We consider an *amortized* model, where the communication and computational costs of the clients reduce with the number of queries. More concretely, while in the setup phase communication and computation is linear in the length of the text, we want that during the query phase the overall communication and the work put by the receivers is *linear in the number of matches* (which is optimal). Of course, we also require the server running in polynomial time. Notice that our strong efficiency requirement comes at a price: it allows the (possibly corrupted) server to learn the number of matches at the very least, as otherwise there is no way to achieve overall communication complexity that is linear in the number of matches. We model this by giving the server *some leakage* for each pattern query which will be described in detail below.

To illustrate the difficulties in designing outsourced protocols for pattern matching let us examine first the simpler keyword search problem, where the goal is to retrieve a record associated with some keyword given that the keyword appears in the database. The keyword search is easily solvable in the outsourced setting by having the simulator commit to a sequence of pairs of ciphertexts, each encrypting (under distinct keys) a pair of keyword and its associated record. Then a search query is simply the secret key of the particular keyword ciphertext. The same solution cannot immediately work for pattern matching as there may be many occurrences of the *same substring*, creating dependency within different text locations (not given to the simulator in advance).

1.1 Our Contribution

In the following we will always talk about a single receiver REC that interacts with a sender SEN and a server S in the query phase. This is only to simplify notation. Our protocols can naturally be applied to a setting with mutually distrusting receivers.

1.1.1 Modeling Outsourced Pattern Matching

We follow the standard method for showing security of protocols using the ideal/real world paradigm [GL90, Bea91, MR91, Can00]. More concretely, in the ideal world the parties just send their inputs over perfectly secure communication channels to a trusted party, who then computes the function honestly and sends the output to the designated party. A real protocol is said to be secure if no adversary can do more harm in a real protocol execution than in an ideal one (where by definition no harm can be done). This definition of security is often called *simulation-based* because security is demonstrated by showing that a real protocol execution can be "simulated" in the ideal world. We give a specification of an ideal execution with a trusted party by defining a reactive outsourced pattern matching functionality $\mathcal{F}_{\mathrm{OPM}}$. This functionality works in two phases: In the preprocessing phase sender SEN uploads its preprocessed text $\widetilde{T}$ to the server. Next, in an iterative query phase, upon receiving a search query p the functionality asks for the approvals of sender SEN (as it may also refuse for this query in the real execution), and the server (as in case of being corrupted it may abort the execution). To model the additional leakage that is required to minimize communication we ask the functionality to forward the server the matched positions in the text upon receiving an approval

from SEN. Note that this type of leakage is necessary if S and REC might collude, as in our case. We further note that our functionality returns all matched positions but can be modified so that only the first few matched positions are returned.[2]

Difficulties with simulating $\mathcal{F}_{\mathrm{OPM}}$. The main challenge in designing a simulator for this functionality is in case when the server is corrupted. In this case the simulator must *commit* to some text in a way that allows it later to produce a sequence of trapdoors that is consistent with the sequence of queries. More precisely, when the simulator commits to a preprocessed text, the leakage that the corrupted server obtains (namely, the positions where the pattern matches the text) has to be consistent with the information that it sees during the query phase. This implies that the simulator must have flexibility when it later matches the committed text to the trapdoors. This difficulty does not arise in the classic two-party setting since there the simulator always plays against a party that contributes an input to the computation which it can first extract, whereas here the server is just a tool to run a computation. Due to this inherent difficulty the text must be *encoded* in a way, that given a search query p and a list of text positions $(i_1, \ldots, i_t)$, one can produce a trapdoor for p in such a way that the "search" in the preprocessed text, using this trapdoor, yields $(i_1, \ldots, i_t)$. We note that alternative (and even simpler) solutions that permute the preprocessed text in order to prevent the server from even learning the matched positions, necessarily require that the server does not collude with the receivers, and even then are not simple to implement. This is because the simulator must have some equivocation mechanism since it cannot distinguish in the preprocessing phase between the last element in a list of matched positions and a non-last element. In contrast, out solutions allow such strong collusion between the server and the receivers.

Searchable/Non-Committing encryption and outsourced pattern matching. To better motivate our solution, let us consider a toy example first. Assume we encrypt each substring of length m in T using searchable encryption [BCOP04], which allows running a search over an encrypted text by producing a trapdoor for the searched word (or a pattern p). Given the trapdoor, the server can check each ciphertext and return the text positions in which the verification succeeds. The first problem that arises with this approach is that searchable encryption does not ensure the privacy of the searched patterns. While this issue may be addressed by tweaking existing constructions of searchable encryption, a more severe problem is that the simulator must commit in advance to (searchable) encryptions of a text that later allow to "find" p at positions that are consistent with the leakage. In other words: all the plaintexts in the specified positions must be associated with the keyword p ahead of time. Of course, as the simulator does not know the actual text T it cannot produce such a consistent preprocessed text. An alternative solution may be given by combining searchable encryption with techniques from non-committing encryptions [CFGN96]. Unfortunately, it is unclear how to combine these two tools even in the random oracle model.

1.1.2 Semi-Honest Outsourced Pattern Matching from Subset Sum

Our first construction for outsourced pattern matching is secure against semi-honest adversaries. In this construction sender SEN generates a vector of random values, conditioned on that the sum of elements in all positions that match the pattern equals some specified value that will be explained below. Namely, SEN builds an instance $\widetilde{T}$ for the *subset sum problem*, where given a trapdoor R the goal is to find whether there exists a subset in $\widetilde{T}$ that sums to R. More formally, the subset sum problem is parameterized by two integers ℓ and M. An instance of the problem is generated by picking random vectors $\widetilde{T} \leftarrow \mathbb{Z}_M^\ell$, $\mathbf{s} \leftarrow \{0,1\}^\ell$ and

[2]This definition is more applicable for search engines where the first few results are typically more relevant, whereas the former variant is more applicable for a DNA search where it is important to find all matched positions. For simplicity we only consider the first variant, our solutions support both variants.

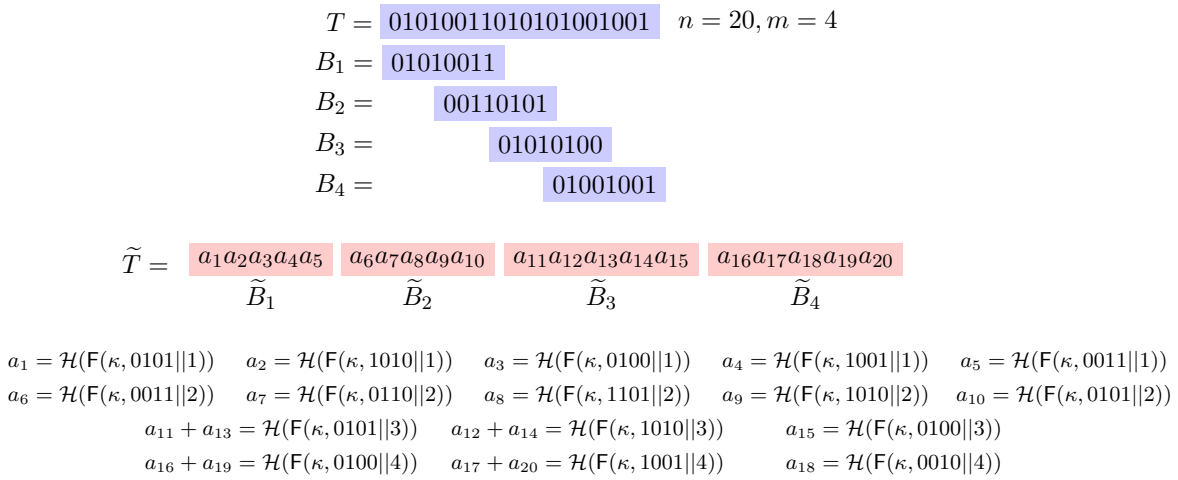

Figure 1: The packaging tecnique applied to a text of length $n = 20$ bits, encoded to search for patterns of length $m = 4$ bits.

outputting $(\widetilde{T}, R = \widetilde{T}^{\mathsf{T}} \cdot \mathbf{s} \bmod M)$. The problem is to find $\mathbf{s}$ given $\widetilde{T}$ and a trapdoor R. Looking ahead, we will have such a trapdoor R_p for each pattern p of length m, such that if p matches T then with overwhelming probability there will be a unique solution to the subset sum instance $(\widetilde{T}, R_p)$. This unique solution is placed at exactly the positions where the pattern appears in the text. The receiver REC that wishes to search for a pattern p obtains this trapdoor from SEN and will hand it to the server. Consequently, we are interested in easy instances of the subset sum problem since we require the server to solve it for each query (we note that using our packaging technique described below, brute force may be applicable as well). This is in contrast to prior cryptographic constructions, e.g., [LPS10] that design cryptographic schemes based on the hardness of this problem. We therefore consider low-density instances which can be solved in polynomial time by a reduction to a short vector in a lattice [LO85, Fri86, CJL+92]. See Figure 1 for an illustration of our idea.

We further note that the security of the scheme relies heavily on the unpredictability of the trapdoor. Namely, in order to ensure that the server cannot guess the trapdoor for some pattern p (and thus solve the subset problem and find the matched locations), we require that the trapdoor is unpredictable. We therefore employ a PRF F on the pattern and fix this value as the trapdoor, where the key k for the PRF is picked by SEN and the two clients SEN and REC communicate via a secure two-party protocol to compute the evaluation of the PRF. This ensures hardness of guessing the trapdoor for any pattern p.

Efficiency considerations. The scheme described above satisfies the appealing properties that the communication complexity during the setup phase is $O(\kappa \cdot n)$ and during the query phase is proportional to the number of matches times κ. Furthermore, the space complexity at the server side is $O(\kappa \cdot n)$. A moment of reflection, however, shows that the scheme has very limited usage in practice. Recall that the server is asked to solve subset sum instances of the form $(\widetilde{T}, R_p)$, where $\widetilde{T}$ is a vector of length $\ell = n - m + 1$ with elements from $\mathbb{Z}_M$ for some integer M. In order to ensure correctness we must guarantee that given a subset sum instance, each trapdoor has a unique solution with high probability. In other words, the collision probability, which equals $2^\ell/M$ (stated also in [IN96]), should be negligible. Fixing $M = 2^{\kappa+n}$ for a security parameter κ, ensures this for large enough κ, say whenever $\kappa \geq 80$. On the other hand, we need the subset sum problem to be solvable in polynomial time. A simple calculation (see analysis in Section 2.2), yields in this case a value of $\ell \approx \sqrt{\kappa}$. This poses an inherent limitation on the length of the text to be preprocessed.

For instance, even using a high value of $\kappa \approx 10^4$ (yielding approximately subset sum elements of size 10 KByte) limits the length of the text to only 100 bits.

An improved solution using packaging. To overcome this limitation, we employ an important extension of our construction based on *packaging*. First, the text is partitioned into smaller pieces of length $2m$ which are handled separately by the protocol, where m is some practical upper bound on the pattern length. Moreover, every two consecutive blocks are overlapping in m positions, so that we do not miss any match in the original text. Even though this approach introduces some overhead, yielding a text T' of overall length $2n$, note that now Eq. (2) below yields $\ell = 2m - m + 1 = m + 1 < \sqrt{\kappa}$, which is an upper bound on the length of the pattern (and not on the length of the text as before). Namely, we remove the limitation on the text length and consider much shorter blocks lengths for the subset sum algorithm.

This comes at a price since we now need to avoid using in each block the same trapdoor for some pattern p, as repetitions allow the server to extract potential valid trapdoors (that have not been queried yet) and figure out information about the text. This is in particular a problem when m is reasonably short as in this case since the server may just try out all potential trapdoors. One may suggest to generate for each block a completely independently chosen trapdoor. Unfortunately, this does not work as during the query phase the receiver needs to communicate these trapdoors to the server which may require communication linear in the length of the text. We solve this problem by requiring from the function outputting the trapdoors to have some form of "programmability" (which allows to simulate the answers to all queries consistently).

Specifically, we implement this function using the random oracle methodology on top of the PRF, so that a trapdoor now is computed by $\mathcal{H}(\mathsf{F}(k,p)\|b)$, for b being the block number. Now, the simulator can program the oracle to match with the positions where the pattern appears in each block. Note that using just the random oracle (without the PRF) is not sufficient as well, since an adversary that controls the server and has access to the random oracle can apply it on p as well. We note that this construction is round optimal, where the number of messages exchanged by the parties is minimal.

1.1.3 Malicious Outsourced Pattern Matching

We extend our construction to the malicious setting as well, tolerating malicious attacks. Our proof ensures that the server returns the correct answers by employing Merkle commitments and zero-knowledge (ZK) sets. Informally speaking, Merkle commitments are succinct commitment schemes for which the commitment size is independent of the length of the committed value (or set). This tool is very useful in ensuring correctness, since now, upon committing to $\widetilde{T}$, the server decommits the solution to the subset sum trapdoor and receiver REC can simply verify that the decommitted values correspond to the trapdoor. Nevertheless, this solution does not cover the case of a mismatch since a corrupted server can always return a "no-match" massage. In order to avoid it we borrow techniques from ZK sets arguments [MRK03], used for proving whether an element is in a specified set without disclosing any further information. Specifically, SEN commits to the set of trapdoors for all patterns p that match T in at least one position. We then ask the server to prove membership/non-membership relative to this set, which contains at most $n - m + 1$ elements. In addition, proving security against a corrupted REC is a straightforward extension of the semi-honest proof using the modifications we made above and the fact that the protocol for implementing the oblivious PRF evaluation is secure against malicious adversaries as well.

The case of a corrupted SEN is more challenging since the simulator needs to extract first the text T, but also verify SEN's computations with respect to the random oracle when it produces $\widetilde{T}$. The only proof technique that we are aware of for proving correctness when using a random oracle is cut-and-choose (e.g., as done in [IKNP03]), which inflates the communication complexity by an additional statistical parameter. Instead, we do not require that the server immediately verifies the correctness of the outsourced text, but

only ensures that if SEN cheats with respect to some query p then it will be caught during the query phase whenever p is queried. The crux of our protocol is that the server does not need to verify all computations at once, but only the computations with respect to the asked queries. This enables us to avoid the costly cut-and-choose technique since verification is done using a novel technique of *derandomizing* SEN's computations. Nevertheless, these difficulties imply that we need to relax the security proof and provide two separated arguments for privacy and correctness whenever the sender is maliciously corrupted instead of a simulation-based security proof. For simplicity, we introduce a separated protocol for each corruption case. A combined protocol can be designed by integrating these into a single protocol.

Efficiency. Our protocols incur higher overhead for the malicious setting due to the use of zero knowledge protocols and Merkle commitments. More specifically, correctness against a corrupted server increases the communication complexity by a factor of $O(\log n)$ in the query phase. This is because each matched position must be verified against its commitment which costs $O(\log n)$ using Merkle commitments. The setup phase is also more costly since the server has to verify the sender's PRF computations which induce communication overhead of $O(\kappa \cdot m \cdot n)$. Note, however, that the long term space complexity is still $O(\kappa \cdot n)$ since the server does not need to store all the ZK proofs for verification. Furthermore, this overhead relies heavily on the [NR97] algebraic PRF complexity; an improved PRF will reduce the overhead complexity.

1.2 Follow-Up Work

In a follow-up work [HZ14], Hazay and Zarosim demonstrated that using the power of the random oracle is essential in order to reduce the resources of receiver REC within round optimal protocols. Specifically, they showed that for certain search functionalities (e.g, pattern matching and all its variants), the communication complexity or the number of steps made by REC is as large as the size of the text in the plain model. Their lower bound applies to both non-private and private channels scenarios (where in the private channels setting corrupted parties do not see the communication between the honest parties). This implies that our semi-honest construction is tight with respect to the assumptions it requires.

2 Preliminaries

2.1 Basic Notations

We let $\mathbb{N}$ be the natural numbers and denote with κ the security parameter. Unless described otherwise, all quantities are implicitly functions of a security parameter denoted $\kappa \in \mathbb{N}$. The security parameter, represented in unary, is an input to all cryptographic algorithms (including the adversary). We let $\mathsf{poly}(\kappa)$ denote an unspecified function $O(\kappa^c)$ for some constant c. A function $\mathsf{negl}(\kappa)$ is *negligible* (in κ) if it is $o(\kappa^{-c})$ for every constant c polynomial. Given a string $a \in \{0,1\}^t$ we specify its value in the ith position by $a[i]$. We write PPT for probabilistic polynomial-time algorithms, i.e., randomized algorithms running in time $\mathsf{poly}(\kappa)$. Let $X = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa\in\mathbb{N}}$ be distribution ensembles. We say that X and Y are computationally indistinguishable, written $X \overset{c}{\approx} Y$, if for any PPT algorithm D we have

$$|\Pr[D(X_\kappa) = 1] - \Pr[D(Y_\kappa) = 1]| \leq \mathsf{negl}(\kappa),$$

where the probability is taken over the random values X_κ and Y_κ, and the randomness of D.

2.2 The Subset Sum Problem

The subset sum problem is parametrized by two integers ℓ and M. An instance of the problem is generated by picking random vectors $\mathbf{a} \leftarrow \mathbb{Z}_M^\ell$, $\mathbf{s} \leftarrow \{0,1\}^\ell$ and outputting $(\mathbf{a}, R = \mathbf{a}^\mathsf{T} \cdot \mathbf{s} \bmod M)$. The problem is to find $\mathbf{s}$ given $\mathbf{a}$ and R. We rely on the following simple observation, which already appeared in [IN96].

Lemma 1 ([IN96]) *Fix ℓ and M. Let $\mathbf{a}$ and $\mathbf{s}$ be chosen uniformly at random and $R = \mathbf{a}^\mathsf{T} \cdot \mathbf{s} \bmod M$. Then, the probability that there exists a vector $\mathbf{s}' \neq \mathbf{s}$ such that $R = \mathbf{a}^\mathsf{T} \cdot \mathbf{s}' \bmod M$ is upper bounded by $2^\ell/M$.*

Proof: It is easy to see that the values $\mathbf{a}^\mathsf{T} \cdot \mathbf{s}$ and $\mathbf{a}^\mathsf{T} \cdot \mathbf{s}'$ are independent and uniformly distributed for every pair $\mathbf{s}, \mathbf{s}'$. Hence,

$$\Pr[\exists\, \mathbf{s} \neq \mathbf{s}' : \mathbf{a}^\mathsf{T} \cdot \mathbf{s} \bmod M = \mathbf{a}^\mathsf{T} \cdot \mathbf{s}' \bmod M] = \sum_{\mathbf{s} \neq \mathbf{s}'} \Pr[\mathbf{a}^\mathsf{T} \cdot \mathbf{s} \bmod M = \mathbf{a}^\mathsf{T} \cdot \mathbf{s}' \bmod M] \leq \frac{2^\ell}{M}. \quad (1)$$

■

The hardness of solving the subset sum problem depends on the ratio between ℓ and $\log M$, which is usually referred to as the *density* Δ of the subset sum instance. In particular:

1. When $\Delta < 1/\ell$, we speak of *low-density* instances which can be solved in polynomial time by a reduction to a short vector in a lattice [LO85, Fri86, CJL+92].

2. When $\Delta > \ell/\log^2 \ell$, we speak of *high-density* instances which can be solved in polynomial time using dynamic programming, or other sophisticated techniques [CFG89, GM91, FP05, Lyu05, Sha08].

In our protocols, we will need to set the parameters in such a way that the subset sum problem is solvable efficiently. Furthermore, we need the term in Eq. (1) to be negligible in the security parameter κ; hence we will set $M = 2^{\kappa+\ell}$. The latter choice immediately rules out algorithms for high-density subset sum (e.g., algorithms based on dynamic programming, since they usually need to process a matrix of dimension M). On the other hand, for low-density instances, Lemma 1 implies $\ell + \kappa > \ell^2$, so that we need to choose κ, ℓ in such a way that

$$\ell < \frac{1}{2}\left(\sqrt{1+4\kappa} - 1\right). \quad (2)$$

Algorithms for solving low-density subset sum are based on lattices. In particular, one can show [CJL+92] that all low-density subset sum instances with $\Delta < 0.9408$ can be solved efficiently with overwhelming probability. The concrete run-time depends on the performances of the LLL algorithm as a function of the lattice dimension; values of $\ell < 1000$ yield to practical performances [CN11, GN08, NS06]. We elaborate more on the impact of the above analysis in our constructions in Section 4.1.

2.3 Collision Resistant Hashing and Merkle Trees

Let in the following $\{\mathcal{H}_\kappa\}_{\kappa\in\mathbb{N}} = \{H : \{0,1\}^{p(\kappa)} \to \{0,1\}^{p'(\kappa)}\}_\kappa$ be a family of hash functions, where $p(\cdot)$ and $p'(\cdot)$ are polynomials so that $p'(\kappa) \leq p(\kappa)$ for sufficiently large $\kappa \in \mathbb{N}$. For a hash function $H \leftarrow \mathcal{H}_\kappa$ a Merkle hash tree [Mer89] is a data structure that allows to commit to $\ell = 2^d$ messages by a single hash value h such that revealing any message requires only to reveal $O(d)$ hash values. A Merkle hash tree is represented by a binary tree of depth d where the ℓ messages $m_1, \ldots, m_\ell$ are assigned to the leaves of the tree; the values assigned to the internal nodes are computed using the underlying hash function H, whereas the value h that commits to $m_1, \ldots, m_\ell$ is assigned to the root of the tree. To open the commitment to a message m_i, one reveals m_i together with all the values assigned to nodes on the path from the root to m_i, and the values

assigned to the siblings of these nodes. We denote the algorithm of committing to ℓ messages $m_1, \ldots, m_\ell$ by $h = \mathsf{Commit}_{\mathsf{M}}(m_1, \ldots, m_\ell)$ and the opening of m_i by $(m_i, \mathrm{path}(i)) = \mathsf{Open}_{\mathsf{M}}(h, i)$. Verifying the opening of m_i is carried out by essentially recomputing the entire path bottom-up and comparing the final outcome (i.e., the root) to the value given at the commitment phase. For simplicity, we abuse notation and denote by $\mathrm{path}(i)$ both the values assigned to the nodes in the path from the root to decommitted value m_i, together with the values assigned to their siblings.

In the paper, we often need to talk about the value assigned to a particular node. To this end, we introduce a labeling scheme for the nodes of a tree. We denote the root of the tree by ε. For a node $w \in \bigcup_{i \leq d}\{0,1\}^i$, we label its left child by $w0$ and its right child by $w1$. The value that is assigned to a node with a label w is typically denoted by h_w. We also consider *incomplete* Merkle trees. An incomplete Merkle tree is a Merkle tree where some nodes w, with $|w| < d$, have *no* leaves. We say that a (possibly incomplete) Merkle tree $\mathcal{T}$ with max depth d is *valid* if for all its nodes w with two children, we have $H(h_{w0}||h_{w1}) = h_w$. We further say that a path $\mathrm{path}(i)$ is *consistent* with a Merkle tree $\mathcal{T}$ (or in $\mathcal{T}$) if all the values assigned to the nodes w in $\mathrm{path}(i)$ are also assigned to the corresponding nodes in $\mathcal{T}$, i.e., $h_w = v_w$, where v_w denotes the value assigned to node w in $\mathrm{path}(i)$.

The standard security property of a Merkle hash tree is collision resistance. Intuitively, this says that it is infeasible to efficiently find a pair (x, x') so that $H(x) = H(x')$, where $H \leftarrow \mathcal{H}_\kappa$ for sufficiently large κ. In fact, one can show that collision resistance of $\{\mathcal{H}_\kappa\}_{\kappa \in \mathbb{N}}$ carries over to the Merkle hashing. Formally, we say that a family of hash functions $\{\mathcal{H}_\kappa\}_\kappa$ is collision resistant if for all PPT adversaries Adv the following experiment outputs 1 with probability $\mathsf{negl}(\kappa)$: (i) A hash function H is sampled from $\mathcal{H}_\kappa$; (ii) The adversary Adv is given H and outputs x, x'; (iii) The experiment outputs 1 if and only if $x \neq x'$ and $H(x) = H(x')$.

2.4 Oblivious Pseudorandom Function Evaluation

Informally speaking, a pseudorandom function (PRF) is an efficiently computable function that looks like a truly random function to any PPT observer. Namely,

Definition 1 (Pseudorandom function) *Let* $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^m \to \{0,1\}^l$ *be an efficient, keyed function. We say* F *is* a pseudorandom function *if for all PPT distinguishers* D*, there exists a negligible function* negl *such that:*

$$|\Pr[D^{\mathsf{F}(k,\cdot)}(1^\kappa) = 1] - \Pr[D^{f_\kappa}(1^\kappa) = 1]| \leq \mathsf{negl}(\kappa),$$

where k *is picked uniformly from* $\{0,1\}^\kappa$ *and* f_κ *is chosen uniformly at random from the set of functions mapping* κ*-bit strings into* l*-bit strings.*

In our protocols, we consider a protocol π_{F} that *obliviously* evaluates a pseudorandom function in the presence of malicious adversaries. Let $k \in \{0,1\}^\kappa$ be a key sampled as above. Then the oblivious PRF evaluation functionality $\mathcal{F}_{\mathrm{PRF}}$ is defined as $(k, x) \mapsto (-, \mathsf{F}(k, x))$. Such an oblivious PRF may be instantiated with the Naor-Reingold pseudorandom function [NR97] that is implemented by the protocol presented in [FIPR05] (and proven in the malicious setting in [HL10]). The function is defined by

$$\mathsf{F}((a_0, \ldots, a_m), x) = g^{a_0 \prod_{i=1}^m a_i^{x[i]}},$$

where g is a generator for a group $\mathbb{G}$ of prime order p, $a_i \in \mathbb{Z}_p$ and $x = (x[1], \ldots, x[m]) \in \{0,1\}^m$.[3] We remark that both the key and the range are not bit strings, as required by Definition 1, but they can be interpreted as such in a natural way. The protocol involves executing an oblivious transfer for every bit of

[3] We remark that this definition considers a function that is not pseudorandom in the classic sense of it being indistinguishable from a random function whose range is composed of all strings of a given length. Rather, it is indistinguishable from a random function whose range is the group generated by g as defined below.

the input x. A two-rounds semi-honest secure implementation can be achieved by using the [FIPR05] protocol combined with any two-rounds semi-honest oblivious transfer. Applying the efficient OT construction of [PVW08] in the malicious setting, implies a two-rounds protocol in the CRS setting with UC security and constant overhead.

2.5 Commitment Schemes

A (non-interactive) commitment scheme consists of a triple of efficient algorithms $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ defined as follows. Upon input the security parameter κ, the probabilistic algorithm Gen outputs a key pk. Upon input the key pk and message $m \in \{0,1\}^*$ (and implicit random coins r), the probabilistic algorithm Commit outputs $(\gamma, \delta) \leftarrow \mathsf{Commit}(\mathsf{pk}, m; r)$ where γ is the committed value, while δ is the decommitment information needed to open the commitment. Typically $\delta = (m, r)$. Upon input the key pk, a message m, and a commitment-pair (γ, δ), the deterministic algorithm Open outputs a bit $b \in \{0,1\}$.

A commitment scheme should be complete, i.e., for any security parameter κ, any $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\kappa)$, for any message $m \in \{0,1\}^*$ and any $(\gamma, \delta) \leftarrow \mathsf{Commit}(\mathsf{pk}, m)$ we have $\mathsf{Open}(\mathsf{pk}, m, \delta, \gamma) = 1$. In addition, commitment schemes are defined by their security properties binding and hiding. Roughly speaking, the binding property says that a sender is unable to change the message it is committed to once the commitment phase is over. The hiding property says that a receiver cannot learn the message from the commitment.

In some cases the public value pk is not needed and the commitment scheme is simply denoted as $(\mathsf{Commit}, \mathsf{Open})$ omitting the algorithm Gen. For ease of notations we omit it from now on. We further require that the commitment scheme is homomorphic. Homomorphic commitment schemes exist based on various hardness assumptions, e.g. the discrete logarithm assumption [Ped91] or decisional Diffie-Hellman (DDH) [ElG85].

2.6 Zero-Knowledge Sets

A zero-knowledge set (ZKS) scheme [MRK03] allows a prover to commit to a secret set G in a way such that it can later prove, non interactively, statements of the form $\gamma \in G$ (or $\gamma \notin G$)), without revealing any further information on G, not even its size. A protocol for ZKS consists of three algorithms $(\mathsf{ZKS\text{-}Setup}, \mathsf{P}, \mathsf{V})$, specified as follows. Algorithm $\mathsf{ZKS\text{-}Setup}$ takes as input the security parameter κ and outputs a common reference string $\mathsf{CRS} \leftarrow \mathsf{ZKS\text{-}Setup}(1^\kappa)$. Algorithm P consists of two sub-algorithms $(\mathsf{P}_1, \mathsf{P}_2)$ such that P_1 takes as input CRS and a set G, and outputs a pair (h_G, σ) where h_G is a commitment to the set G and σ is some state information; P_2 takes as input the state information σ and a value γ, and outputs a proof π. Algorithm V takes as input CRS, a commitment h_G, some value γ and a proof π, and outputs a decision bit.

Intuitively, we require the following properties to hold. (1) Completeness: For any set G, for any γ such that $\gamma \in G$ (resp. $\gamma \notin G$) an honest prover who correctly commits to G can always convince the verifier that $\gamma \in G$ (resp. $\gamma \notin G$); (2) Soundness: Once a commitment to the set G has been formed (even by a malicious prover), no P can, for the same γ, convince the verifier that both $\gamma \in G$ and $\gamma \notin G$; (3) Zero-knowledge: There exists a simulator $\mathsf{Sim}_{\mathrm{ZKsets}}$ such that even for adversarially chosen G, no adversarial verifier can tell whether it is (a) talking to an honest prover P committed to G, or (b) talking to $\mathsf{Sim}_{\mathrm{ZKsets}}$ who only has oracle access to the set G. We refer the reader directly to [MRK03] for formal definitions.

The original construction of zero-knowledge sets is based on special properties of a commitment scheme (so called *mercurial commitments*) which have been formalized later on by Chase *et al.* [CHL$^+$05]. All known constructions of ZKS are built upon the common idea of constructing an authenticated Merkle tree of depth κ where each internal node is a mercurial commitment of its two children. For a universe of size 2^κ fix a security parameter l; in particular $l = 256$ suffices to get $\kappa = 128$ bits of security. The most efficient proofs relying on mercurial commitments requires $6\kappa + 5$ elements (for proofs of membership) and $5\kappa + 4$ elements (for proofs of non-membership), where each element has size l bits. An improvement

can be obtained relying on q-mercurial commitments [CRFM11], where a proof of membership requires $\mu(q+4)+5$ elements and a proof of non-membership requires $4\mu+4$ elements, where now the size of the universe is q^μ. A value of $q = 8$ yields proofs of membership that are 33% shorter and proofs of non-membership that are almost 73% shorter.

3 Modeling Outsourced Pattern Matching

The inputs for the basic pattern matching problem are a text T of length n and a pattern p (i.e., keyword) of length m; the goal is to find all the text locations in which the pattern matches the text. A private distributed variant of this problem is defined in the two-party setting, where a sender SEN holds a text T and a receiver REC holds a pattern p. The goal of REC is to learn the positions in which p matches in the text, without revealing anything about the pattern to SEN; at the same time, REC should not learn anything else about the text.[4] In this section we are interested in an *outsourced* variant of the problem, which is specified in two phases. In the *setup phase* a sender SEN uploads a (preprocessed) text $\widetilde{T}$ to an external server S. This phase is run only once. In the *query phase* receivers $\mathrm{REC}_1, \mathrm{REC}_2, \ldots$ query the text by searching patterns and learn the matched text locations. For simplicity, we focus on a single receiver REC asking multiple queries. However, our model can be easily generalized to the multiple receivers scenario.

The basic idea is to implement the pattern matching functionality using the server as a mediator, answering search queries on behalf of SEN. In order to take some advantage from this modeling, we must allow the server to obtain some leakage about the text, otherwise the communication complexity between the server and receiver REC would be $O(n)$. Instead, we are interested in building schemes where the preprocessing phase requires $O(n)$ workload, but the overall cost of issuing a query grows only linearly with the number of matches (which is as optimal as one can obtain). This optimization comes with the price of revealing some leakage about the text. More precisely, the server learns that for some text positions repetitions occur. Attempts to hide this information from the sever by permuting the text fail if the server colludes with REC. For some applications this leakage is tolerable given the improvement of running search queries. To sum up, we aim for $O(n)$ computation/communication overhead in the preprocessing phase and $O(t_p)$ in the query phase, where t_p is the number of occurrences of p in T.

We further require that the round complexity of any protocol implemented in this setting is minimal. That is, in the setup phase we require a single message sent from the sender to the server, whereas in the query phase we require clients REC and SEN to exchange only two messages (one in each direction), and one message in each direction between REC and S in order to retrieve the output. We note that our constructions meet this order of rounds, but this may not be the case in general. We denote a scheme with this number of rounds by *round optimal*.[5]

We formalize security using the ideal/real paradigm. Note that, in the context of outsourced computation, the server is a separate entity that does not contribute any input to the computation and is required to run most of the function evaluation. In the ideal setting, such an entity is also communicating with the functionality and, upon corruption, decides whether the functionality sends the outcome of the computation to the prescribed receivers. Denote by T_j the substring of length m that starts at text location j. The pattern matching ideal functionality in the outsourced setting is depicted in Figure 2. We write $|T|$ for the bit length of T and assume that receiver REC asks a number of queries p_i ($i \in [\lambda]$).

[4] As specified in the introduction, we can define a search functionality that only returns the first few and most relevant matches. Our discussion and formalization below can be easily adapted for this functionality as well.

[5] Looking ahead, we manage to meet this optimal bound only for our semi-honest protocol.

Functionality $\mathcal{F}_{\mathrm{OPM}}$

Let $m, \lambda \in \mathbb{N}$. Functionality $\mathcal{F}_{\mathrm{OPM}}$ sets an empty table $\mathcal{B}$ and proceeds as follows, running with clients SEN and REC, server S and adversary Sim.

1. Upon receiving a message (text, T, m) from SEN, send $(\mathsf{preprocess}, |T|, m)$ to S and Sim, and record (text, T).
2. Upon receiving a message (query, p_i) from receiver REC (for $i \in [\lambda]$), where message $(\mathsf{text}, \cdot)$ has been recorded and $|p_i| = m$, it checks if the table $\mathcal{B}$ already contains an entry of the form $(p_i, \cdot)$. If this is not the case then it picks the next available identifier id from $\{0,1\}^*$ and adds (p_i, id) to $\mathcal{B}$. It sends $(\mathsf{query}, \mathrm{REC})$ to SEN and Sim.
   (a) Upon receiving $(\mathsf{approve}, \mathrm{REC})$ from sender SEN, read (p_i, id) from $\mathcal{B}$ and send $(\mathsf{query}, \mathrm{REC}, (i_1, \ldots, i_t), \mathsf{id})$ to server S, for all text positions $\{i_j\}_{j \in [t]}$ such that $T_{i_j} = p_i$. Otherwise, if no $(\mathsf{approve}, \mathrm{REC})$ message has been received from SEN, send $\perp$ to REC and abort.
   (b) Upon receiving $(\mathsf{approve}, \mathrm{REC})$ from Sim, read (p_i, id) from $\mathcal{B}$ and send $(\mathsf{query}, p_i, (i_1, \ldots, i_t), \mathsf{id})$ to receiver REC. Otherwise, send $\perp$ to receiver REC.

Figure 2: The outsourced pattern matching functionality

The definition. As in the standard static modeling, a corrupted party is either passively or actively controlled by an adversarial entity. In the passive case (a.k.a. semi-honest case) a corrupted party follows the protocol's instructions and tries to gain additional information about the honest parties' inputs from its view; in the active case (a.k.a. malicious case) a corrupted party is allowed to follow an arbitrary polynomial-time strategy. In our case, when the server is corrupted we must ensure that the only information leaked by the protocol is about the text positions for which repetitions occur, without disclosing the actual content of the text in these positions or any additional information. A moment of reflection shows that in the security proof the simulator needs to commit to the text *before* given the above leakage from the trusted party. We emphasize that this technicality is not artificial, since even if receiver REC is corrupted at the beginning of the execution, the simulator cannot be given the leakage about the queries in advance since the queries may be asked in an fully adaptive manner. In other words, it may be the case that receiver REC does not know all the queries it will ask in advance.

Formally, denote by $\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}}, \mathsf{Sim}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ the output of an ideal adversary Sim, server S and clients SEN, REC in the above ideal execution of $\mathcal{F}_{\mathrm{OPM}}$ upon inputs $(-, (T, (p_1, \ldots, p_\lambda)))$ and auxiliary input z given to Sim. We note that one can also consider a security definition that captures collusion between the server and one of the clients. Our protocols capture collusion between S and REC. We implement functionality $\mathcal{F}_{\mathrm{OPM}}$ via three two-party protocols $\pi = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$ specified as follows. Protocol π_{Pre} is run in the preprocessing phase by SEN to preprocess text T and forwards the outcome $\widetilde{T}$ to S. During the query phase, protocol π_{Query} is run between SEN and REC (holding a pattern p); this protocol outputs a trapdoor R_p that depends on p and will enable the server to search the preprocessed text. Lastly, protocol π_{Opm} is run by S upon input the preprocessed text and a trapdoor (forwarded by REC); this protocol returns to REC the matched text positions (if any). We denote by $\mathbf{REAL}_{\pi, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ the output of adversary Adv, server S and clients SEN, REC in a real execution of $\pi = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$ upon inputs $(-, (T, (p_1, \ldots, p_\lambda)))$ and auxiliary input z given to Adv.

Definition 2 (Security of outsourced pattern matching) *We say that* π securely implements $\mathcal{F}_{\mathrm{OPM}}$*, if for any PPT real adversary* Adv *there exists a PPT ideal adversary (simulator)* Sim *such that for any tuple of*

inputs $(T, (p_1, \ldots, p_\lambda))$ *and auxiliary input* z*,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa\in\mathbb{N}} \stackrel{\mathrm{c}}{\approx} \{\mathbf{REAL}_{\pi,\mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa\in\mathbb{N}}.$$

The schemes described in the next sections implement the ideal functionality $\mathcal{F}_{\mathrm{OPM}}$ in the random oracle model. As usual in this case, we assume the existence of a publicly available function H behaving as a truly random function. We stress that all parties are allowed to query the random oracle arbitrarily, regardless the fact that they are malicious or semi-honest. In addition, we describe our constructions in a private channels setting, where the corrupted party is not allowed to see the communication between the honest parties. This is because our protocol does not support a collusion between the sender and the server. Thus, if a corrupted sender sees the receiver's message, the receiver's privacy is violated. Note that any protocol in this setting can be transformed into the non-private channels setting using PKI (namely, by having every pair of parties encrypt their communication using a pair of public keys). This, however, requires additional rounds of communication.

4 Security in the Presence of Semi-Honest Adversaries

In this section we present our implementation of the outsourced pattern matching functionality $\mathcal{F}_{\mathrm{OPM}}$ and prove its security against semi-honest adversaries. A scheme with security against malicious adversaries is described in Section 5, building upon the protocol in this section. Recall first that in the outsourced variant of the pattern matching problem, sender SEN manipulates the text T and then stores it on the server S in such a way that the preprocessed text can be used later to answer search queries submitted by receiver REC. The challenge is to find a way to hide the text (in order to obtain privacy), while enabling the server to carry out searches on the hidden text whenever it is in possession of an appropriate trapdoor.

We consider a new approach and reduce the pattern matching problem to the subset sum problem (cf. Section 2.2). Namely, consider a text T of length n, and assume we want to allow searches for patterns of length m. For some integer $M \in \mathbb{N}$, we assign to each distinct pattern p that appears in T a random element $R_p \in \mathbb{Z}_M$. Letting $\ell = n - m + 1$, the preprocessed text $\widetilde{T}$ is a vector in $\mathbb{Z}_M^\ell$ with elements specified as follows. Specifically, for each pattern p that appears t times in T, we sample random values $a_1, \ldots, a_t \in \mathbb{Z}_M$ such that $R_p = \sum_{j=1}^{t} a_j$. Denote with $i_j \in [\ell]$ the jth position in T where p appears and set $\widetilde{T}[i_j] = a_j$. Notice that for each pattern p, there exists a vector $\mathbf{s} \in \{0,1\}^\ell$ such that $R_p = \widetilde{T}^{\mathsf{T}} \cdot \mathbf{s}$. Hence, the positions in $\widetilde{T}$ where pattern p matches are identified by a vector $\mathbf{s}$ and can be viewed as the solution for the subset sum problem instance $(R_p, \widetilde{T})$.

Roughly, our protocol works as follows. During protocol π_{Pre}, we let the sender SEN generate the preprocessed text $\widetilde{T}$ as described above, and send the result to the server S. Later, when a receiver REC wants to learn at which positions a pattern p matches the text, clients REC and SEN run protocol π_{Query}; at the end of this protocol REC learns the trapdoor R_p corresponding to p. Finally, during π_{Opm} receiver REC sends this trapdoor to S, which can solve the subset sum problem instance $(R_p, \widetilde{T})$. The solution to this problem corresponds to the matches of p, which are forwarded to the receiver REC. To avoid that SEN needs to store all trapdoors, we rely on a PRF to generate the trapdoors itself. More precisely, instead of sampling the trapdoors R_p uniformly at random, we set $R_p := \mathsf{F}(k, p)$, where F is a PRF. Thus, during the query phase REC and SEN run an execution of an oblivious PRF protocol; at the end of this protocol REC learns the output of the PRF, i.e., the trapdoor R_p.

Although the protocol described above provides a first basic solution for the outsourced pattern matching, it suffers from a strong restriction as only very short texts are supported. We will provide more details

Protocol $\pi_{\mathsf{SH}} = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$

Let $\kappa \in \mathbb{N}$ be the security parameter and let M, m, n, μ be integers, where for simplicity we assume that n is a multiple of $2m$. Further, let $\mathcal{H} : \{0,1\}^{\mu} \to \mathbb{Z}_M$ be a random oracle and $\mathsf{F} : \{0,1\}^{\kappa} \times \{0,1\}^{m} \to \{0,1\}^{\mu}$ be a PRF. Protocol π_{SH} involves a sender SEN holding a text $T \in \{0,1\}^{n}$, a receiver REC querying for patterns $p \in \{0,1\}^{m}$, and a server S. The interaction between the parties is specified below.

Setup phase, π_{Pre}**.** The protocol is invoked between sender SEN and server S. Given input T and integer m, sender SEN picks a random key $k \in \{0,1\}^{\kappa}$ and prepares first the text T for the packaging by writing it as

$$T' := (B_1, \ldots, B_u) = ((T[1], \ldots, T[2m]), (T[m{+}1], \ldots, T[3m]), \ldots, (T[n{-}2m{+}1], \ldots, T[n])),$$

where $u = n/m - 1$. Next, for each block B_b and each of the $m+1$ patterns $p \in \{0,1\}^{m}$ that appear in B_b we proceed as follows (suppose there are at most t matches of p in B_b).

1. Sender SEN evaluates $R_p := \mathcal{H}(\mathsf{F}(k,p)||b)$, samples $a_1, \ldots, a_{t-1} \in \mathbb{Z}_M$ at random and then fixes a_t such that $a_t = R_p - \sum_{j=1}^{t-1} a_j \bmod M$.
2. Set $\widetilde{B}_b[v_j] = a_j$ for all $j \in [t]$ and $v_j \in [m+1]$. Note that here we denote by $\{v_j\}_{j\in[t]}$ ($v_j \in [m+1]$) the set of indexes corresponding to the positions where p occurs in B_b. Later in the proof we will be more precise and explicitly denote to which block v_j belongs by using explicitly the notation v_{j_b}.

Finally, SEN outsources the text $\widetilde{T} = (\widetilde{B}_1, \ldots, \widetilde{B}_u)$ to S.

Query phase, π_{Query}**.** Upon issuing a query $p \in \{0,1\}^{m}$ by receiver REC, clients SEN and REC engage in an execution of protocol π_{Query} which implements the oblivious PRF functionality $(k,p) \mapsto (-, \mathsf{F}(k,p))$. Upon completion, REC learns $\mathsf{F}(k,p)$.

Oblivious pattern matching phase, π_{Opm}**.** This protocol is executed between server S (holding $\widetilde{T}$) and receiver REC (holding $\mathsf{F}(k,p)$). Upon receiving $\mathsf{F}(k,p)$ from REC, the server proceeds as follows for each block $\widetilde{B}_b$. It interprets $(\widetilde{B}_b, \mathcal{H}(\mathsf{F}(k,p)||b))$ as a subset sum instance and computes $\mathbf{s}$ as the solution of $\widetilde{B}_b \cdot \mathbf{s} = \mathcal{H}(\mathsf{F}(k,p)||b)$. Let $\{v_j\}_{j\in[t]}$ denote the set of indexes such that $\mathbf{s}[v_j] = 1$, then the server S returns the set of indexes $\{\varphi(b, v_j)\}_{b\in[u], j\in[t]}$ to the receiver REC.

Figure 3: Semi-honest outsourced pattern matching

on this restriction in Section 4.1.[6] To overcome this severe limitation, we partition the text into smaller pieces each of length $2m$, where each such piece is handled as a separate instance of the the protocol. More specifically, for a text $T = (T[1], \ldots, T[n])$ let $(T[1], \ldots, T[2m]), (T[m+1], \ldots, T[3m]), \ldots$ be blocks, each of length $2m$, such that every two consecutive blocks overlap in m bits. Then, for each pattern p that appears in the text the sender SEN computes an individual trapdoor for each block where the pattern p appears. More precisely, suppose that pattern p appears in block B_b then we compute the trapdoor for this block (and pattern p) as $\mathcal{H}(\mathsf{F}(k,p)||b)$. Here, $\mathcal{H}$ is a cryptographic hash function that will be modeled as a random oracle in our proofs. Given the trapdoors, we apply the preprocessing algorithm to each block individually. The sub-protocols π_{Query} and π_{Opm} work as described above with a small change. In π_{Query} receiver REC learns the output of the PRF $\mathsf{F}(k,p)$ instead of the actual trapdoors and in π_{Opm} receiver REC forwards directly the result $\mathsf{F}(k,p)$ to S. The server can then compute the actual trapdoor using the random oracle. This is needed to keep the communication complexity of the protocol low. Note that in this case if

[6] Taking a look ahead, this solution can only support short texts as otherwise either the collision probability (cf. Lemma 1) is large and we cannot achieve correctness, or the subset sum problem cannot be solved efficiently as the instance is not low-density.

we let $\{v_{j_b}\}_{j_b \in [t_b]}$ be the set of indices corresponding to the positions where p occurs *in a given block* B_b, the server needs to map these positions to the corresponding positions in T (and this has to be done for each of the blocks where p matches). It is easy to see that such a mapping from a position v_{j_b} in block B_b to the corresponding position in the text T can be computed as $\varphi(b, v_j) = (b-1)m + v_j$. The entire protocol, including the packaging mechanism, is shown in Figure 3; see also Figure 1 for a pictorial representation.

We now prove the following result.

Theorem 1 *Let $\kappa \in \mathbb{N}$ be the security parameter. For integers n, m we set $\lambda = \mathsf{poly}(\kappa), \mu = \mathsf{poly}(\kappa)$, $u = n/m - 1$, $\ell = (m+1)u$ and $M = 2^{m+\kappa+1}$. We furthermore require that κ is such that $2^{m+1}/M$ is negligible (in κ). Assume $\mathcal{H} : \{0,1\}^{\mu} \to \mathbb{Z}_M$ is a random oracle and $\mathsf{F} : \{0,1\}^{\kappa} \times \{0,1\}^{m} \to \{0,1\}^{\mu}$ is a pseudorandom function. Then, round optimal protocol π_{SH} from Figure 3 securely implements the $\mathcal{F}_{\mathrm{OPM}}$ functionality in the presence of semi-honest adversaries.*

It is easy to verify that the number of rounds within π_{SH} is optimal. First, SEN sends only one message to S. Next, we consider a two-rounds oblivious PRF evaluation protocol for π_{Query}. Finally, REC and S exchange only two messages. We now continue with our security proof.

Proof: We first argue about correctness and then prove privacy.

Correctness. We say that our construction achieves correctness if with overwhelming probability each pattern query p issued by REC is answered correctly with respect to the outsourced text T. More concretely, for each pattern $p \in \{0,1\}^m$ and a text $T \in \{0,1\}^n$, let $\{i_j\}_{j \in [t]}$ be the positions in T where p matches the text. Then, our protocol achieves correctness if for any T and p it returns the correct matches $\{i_j\}_{j \in [t]}$. Suppose that pattern p appears at position i_j. We need to show that in this case the algorithm given in Figure 3 returns indeed index i_j. To this end, suppose that position i_j lies in block B_b and B_{b+1} for some $b \in [1, v-1]$ (recall that two consecutive blocks overlap at m positions, hence the bit $T[i_j]$ may appear in both blocks; in case we have a match of a pattern exactly in the area that overlaps, then we will always consider only the match in the first block). Wlog. assume that pattern p appears in block B_{b+1}. We run the preprocessing algorithm on block B_{b+1} to obtain the transformed block $\widetilde{B}_{b+1}$ that contains the solution of the subset sum problem for $(\mathcal{H}(\mathsf{F}(k,p)||b+1), \widetilde{B}_{b+1})$ at positions where the pattern p appears. Hence, when during the execution of protocol π_{Opm} the server solves this subset sum instance, it will retrieve index i_j as one of the solutions.

The analysis from above does not hold when one of the following situations occur. First, it may be the case that for two different patterns $p \neq p'$ we get a collision in the PRF and/or random oracle. The probability that this happens is negligible by the birthday bound. Next, we consider the following two cases when we get a non-unique solution for the subset sum problem:

1. *Two (or more) different subsets that sum to the same trapdoor in some block:* From Lemma 1, it follows that for each subset sum instance collision happens with probability $2^{2m}/M$. Taking the union bound we get that the probability of collision for all patterns p (appearing in some block of length $2m$) is upper bounded by $2m \cdot 2^{2m}/M$, which for our choice of parameters is negligible in κ.

2. *There is no match in a block, but a subset in this block sums to trapdoor:* For each trapdoor R_p the probability that some subset in a block sums to R_p is upper bounded by $1/M$. As there are 2^m possible targets, we get by the union bound that the probability of this event is upper bounded by $2^m/M$ which is negligible in κ.

As both events above occur with a negligible probability and there are $u = n/m - 1$ blocks, we can apply the union bound, resulting in a negligible (in κ) probability of an incorrect result for our protocol. This concludes the correctness proof of our protocol.

Privacy. We will show that for any PPT real adversary Adv there exists a PPT ideal adversary (simulator) Sim such that for any tuple of inputs $(T, (p_1, \ldots, p_\lambda))$ and auxiliary input z,

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{REAL}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa\in\mathbb{N}}.$$

We prove each corruption setting separately. The final simulator from above can then be obtained by combining the individual simulators.

A corrupted server. We begin with the case when the server is corrupted. Let Adv denote the adversary controlling the server S. We will build a simulator $\mathsf{Sim}_{\mathrm{S}}$ that simulates the view of Adv. To this end $\mathsf{Sim}_{\mathrm{S}}$ interacts with the trusted party and uses the leakage from the trusted party, which for each pattern query reveals the matched text positions. We first describe the simulator $\mathsf{Sim}_{\mathrm{S}}$ with access to Adv.

> **Convention:** During the simulation $\mathsf{Sim}_{\mathrm{S}}$ evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by Adv or by $\mathsf{Sim}_{\mathrm{S}}$ during its simulation of the corrupted server. To evaluate $\mathcal{H}(x)$, $\mathsf{Sim}_{\mathrm{S}}$ first checks if it has already recorded a pair (x, r), in which case $\mathcal{H}(x)$ evaluates to the value r. Otherwise, $\mathsf{Sim}_{\mathrm{S}}$ chooses a random string $r \in \mathbb{Z}_M$, records (x, r) and evaluates $\mathcal{H}(x)$ to r.

1. On input the auxiliary input z, $\mathsf{Sim}_{\mathrm{S}}$ invokes Adv (i.e., the corrupted server) on this input. The simulator keeps track of a table $\mathcal{B}$ that is initially set to the empty table.

2. Upon receiving a $(\mathsf{preprocess}, n, m)$ message from the trusted party denoting that the honest SEN wants to outsource a text of length n to the trusted party, $\mathsf{Sim}_{\mathrm{S}}$ defines text $\widetilde{T}$ by sampling uniformly at random a vector of length $\ell := (m+1)u$ from $\mathbb{Z}_M^\ell$. It forwards $\widetilde{T}$ to adversary Adv and stores it for later usage as well.

3. Upon receiving a $(\mathsf{query}, \mathrm{REC}, (i_1, \ldots, i_t), \mathsf{id})$ message from the trusted party indicating that receiver REC submitted a search query that was approved by SEN, $\mathsf{Sim}_{\mathrm{S}}$ distinguishes two cases.

   (a) *Pattern queried by* REC *appears in* T*:* In this case $\{i_j\}_{j\in[t]}$ is not the empty set, and the simulator samples uniformly at random a value X_{id} from $\{0,1\}^\mu$ (this will take the role of $\mathsf{F}(k,p)$ in the real execution). Then, it proceeds as follows for each of the i_j's. It first computes the block number $b = \lfloor i_j/m \rfloor + 1$ in which the index i_j occurs and then the starting position $v_{j_b} = i_j \bmod m$ where the pattern appears in $\widetilde{B}_b$. Then, $\mathsf{Sim}_{\mathrm{S}}$ programs the random oracle $\mathcal{H}(X_{\mathsf{id}}||b)$ to $\sum_{j_b=1}^{t_b} \widetilde{T}[v_{j_b}]$; if $\mathcal{H}$ has already been programmed to a different value, then we abort.

   (b) *Pattern does not appear in* T*:* In this case $\{i_j\}_{j\in[t]}$ is the empty set, and we check if table $\mathcal{B}$ contains a value of the form $(\mathsf{id}, X_{\mathsf{id}})$. Otherwise, we pick the value X_{id} uniformly at random in $\{0,1\}^\mu$ and store $(\mathsf{id}, X_{\mathsf{id}})$ in $\mathcal{B}$.

   Finally, $\mathsf{Sim}_{\mathrm{S}}$ (emulating the role of REC in the real execution) forwards X_{id} to the adversary.

4. If Adv does not answer with $(i_1, \ldots, i_t)$, $\mathsf{Sim}_{\mathrm{S}}$ sends $\perp$ to ideal functionality and abort. Otherwise it sends the trusted party $(\mathsf{approve}, \mathrm{REC})$.

5. $\mathsf{Sim}_{\mathrm{S}}$ outputs whatever Adv does.

We first note that $\mathsf{Sim}_{\mathrm{S}}$ runs in polynomial time since it only samples a random vector from $\mathbb{Z}_M^\ell$, and then calculates the sum of values from a given subset. Next, we show that the distribution produced by $\mathsf{Sim}_{\mathrm{S}}$ in the ideal world is computationally indistinguishable from the distribution that Adv expects to see in the real

world. This is required to hold even given the leakage revealing the positions where the pattern matches the text. We start by defining a hybrid distribution $\mathbf{HYB}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ that is defined as the real experiment $\mathbf{REAL}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ with the difference that X_{id} is not computed by a PRF but rather by a random function f_κ. The following claim formalizes this statement.

Claim 2 *Let* F *be a secure pseudorandom function (cf. Definition 1), then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for sufficiently large* $\kappa \in \mathbb{N}$ *and for any tuple of inputs* $(T,(p_1,\ldots,p_\lambda))$ *and auxiliary input* z*, it holds that*

$$\{\mathbf{REAL}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{HYB}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}. \quad (3)$$

The proof follows by an easy reduction to the pseudorandomness of F, as a distinguisher for the distributions in Eq. (3) yields a distinguisher for the PRF.

To move to the simulated view, we need to bound the distance between the experiment $\mathbf{HYB}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ and the simulated view. To this end, we define the event bad that occurs when the simulator aborts in the ideal world.

- Event bad: Occurs if the simulation given above is aborted. In this case the corrupted server has asked for a direct query to the random oracle of the form $(f_\kappa(p)||b)$ *before* it has seen $f_\kappa(p)$, where $b\in[u]$ and p is a pattern that occurs in the text T. Recall that f_κ is a random function as defined in the hybrid world.

We will show that the distribution produced by the simulator Sim_S in the ideal world is statistically close to the distribution produced in the hybrid world.

Claim 3 *For any input text* T*, patterns* $p_1,\ldots,p_\lambda$*, and auxiliary input* z*, it holds that*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}} \equiv_s \{\mathbf{HYB}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}.$$

Proof: We show that the view generated by Sim_S for the corrupted server in the ideal world has the same distribution as the server expects to see in the hybrid protocol conditioned on the event $\overline{\mathsf{bad}}$. The view contains the preprocessed text, answers to random oracle queries and the trapdoors that are sent by REC during the sub-protocol π_{Opm}. In the ideal world, Sim_S chooses these values in the following way.

- *Answers to direct RO queries:* If the RO has already been asked on this input, then it returns the stored value; otherwise it returns a value chosen uniformly at random,
- *Preprocessed text:* Sampled independently and uniformly from $\mathbb{Z}_M^\ell$,
- *Trapdoors sent by* REC*:* We first check if table $\mathcal{B}$ contains an entry of the form $(\mathsf{id},X_{\mathsf{id}})$, in which case we return X_{id}. Otherwise, we consider two cases depending on the query $(\mathsf{query},\mathrm{REC},(i_1,\ldots,i_t),\mathsf{id})$:
  1. *Pattern matches the text:* X_{id} is chosen uniformly at random from $\{0,1\}^\mu$. For each block B_b where the pattern appears, we program the random oracle $\mathcal{H}(X_{\mathsf{id}}||b)$ to $\sum_{j_b=1}^{t_b}\widetilde{T}[v_{j_b}]$, where v_{j_b} are the positions in block $\widetilde{B}_b$ where the pattern appears. We store $(\mathsf{id},X_{\mathsf{id}})$ in $\mathcal{B}$.
  2. *Pattern does not match the text:* We pick X_{id} at random from $\{0,1\}^\mu$ and store $(\mathsf{id},X_{\mathsf{id}})$ in B.

  Notice that in the first case the trapdoor is a uniformly chosen value from $\mathbb{Z}_M$ as it is the sum of uniformly and independently chosen values.

It is easy to see that individually these values are identically distributed in both the ideal and hybrid execution. For the proof, we need to analyze the joint distribution that the corrupted server sees. The only difference between the joint distribution in the ideal world and in the hybrid world is the way in which the preprocessed text and the trapdoors are sampled. While in the ideal world the preprocessed text is sampled uniformly and independently from $\mathbb{Z}_M^\ell$ in the hybrid world we prepare it according to the patterns that appear in the text T. More precisely, in the hybrid world we put at locations where a pattern appears a random additive sharing of a random value. This trivially implies that also in the hybrid world the transformed text is sampled uniformly at random from $\mathbb{Z}_M^\ell$. It remains to argue that at the later stage when the receiver REC asks for patterns, the view in the ideal world remains consistent, namely, the sum of the values that are put at the matched positions is equal to the trapdoor. We can achieve this consistency by programming the value of the random oracle to the appropriate value. There is one exception when such programming fails: namely, when the adversary has earlier queried the random oracle on this value. This is exactly when event bad happens and the simulator aborts. It remains to show that the probability that event bad happens is negligible in the security parameter.

Event bad occurs when the adversary asks the random oracle on a value X with form $(f_\kappa(p)||b)$ for some $b \in [u]$ and pattern p that appears in the text *before* it actually sees X. As f_κ is a random function this happens with probability at most $\mathsf{poly}(\kappa)/2^\mu = \mathsf{negl}(\kappa)$. This concludes the proof. ■

Combining Claims 2 and 3, it holds that the distributions $\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}$ and $\{\mathbf{REAL}_{\pi_{\mathsf{SH}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}$ are computationally close, which concludes the proof of privacy in case of a corrupted (semi-honest) server.

A corrupted sender. Next, we consider the case when SEN is corrupted. Let Adv denote an adversary controlling sender SEN, we build a simulator $\mathsf{Sim}_{\mathrm{SEN}}$ that generates its view. The simulator $\mathsf{Sim}_{\mathrm{SEN}}$ needs to emulate the roles of receiver REC and of the server S using the leakage it gets from the trusted party. Let us describe first the simulator $\mathsf{Sim}_{\mathrm{SEN}}$ that is given access to adversary Adv.

> **Convention:** During the simulation $\mathsf{Sim}_{\mathrm{SEN}}$ evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by Adv or by $\mathsf{Sim}_{\mathrm{SEN}}$ during its simulation of the corrupted SEN. To evaluate $\mathcal{H}(x)$, $\mathsf{Sim}_{\mathrm{SEN}}$ first checks if it has already recorded a pair (x,r), in which case $\mathcal{H}(x)$ evaluates to the value r. Otherwise, $\mathsf{Sim}_{\mathrm{SEN}}$ chooses a random string $r \in \mathbb{Z}_M$, records (x,r) and evaluates $\mathcal{H}(x)$ to r.

1. On input text T, length m, and auxiliary input z, $\mathsf{Sim}_{\mathrm{SEN}}$ invokes Adv on these inputs.
2. Upon receiving $\widetilde{T}$ from Adv, $\mathsf{Sim}_{\mathrm{SEN}}$ sends a (text, T, m) message to the trusted party and receives back $(\mathsf{preprocess}, |T|, m)$.
3. Upon receiving a $(\mathsf{query}, \mathrm{REC})$ message from the trusted party, denoting that the honest REC submitted a query p of length m to the trusted party, $\mathsf{Sim}_{\mathrm{SEN}}$ invokes the simulator $\mathsf{Sim}_{\mathsf{Query}}$ for protocol π_{Query} to simulate Adv's view. If during the simulation $\mathsf{Sim}_{\mathsf{Query}}$ sends $\perp$ to its own trusted party, $\mathsf{Sim}_{\mathrm{SEN}}$ sends also $\perp$ to the trusted party, aborting the execution. Otherwise, it sends $(\mathsf{approve}, \mathrm{REC})$ to the trusted party on behalf on SEN as well (we recall that the functionality excepts to receive an approval from both SEN and the ideal adversary).
4. $\mathsf{Sim}_{\mathrm{SEN}}$ outputs whatever Adv outputs.

We first note that the simulator runs in polynomial time since all it does is running the polynomial time simulator $\mathsf{Sim}_{\mathsf{Query}}$ for protocol π_{Query}. Next, we analyze the privacy of receiver REC and show that SEN does not gain any further information during the execution of the real protocol π_{SH}. In fact, the only difference between the simulation above and a real execution of π_{SH} is in the way the view of the corrupted

SEN is generated with respect to a query p which has been approved by SEN. Namely, in the simulation this view is produced by the simulator $\mathsf{Sim}_{\mathsf{Query}}$ whereas in a real execution it is produced by a run of π_{Query}. Indistinguishability of the two distributions thus follows from the fact that π_{Query} securely implements the oblivious PRF functionality.

A corrupted receiver. Finally, we consider the case when receiver REC is corrupted. Let Adv denote an adversary controlling receiver REC, we build a simulator Sim_C that generates its view. Simulator $\mathsf{Sim}_{\mathrm{REC}}$ needs to emulate the roles of the sender SEN and of the server S using the leakage it gets from the trusted party. Below we describe the simulator Sim_C that is given access to adversary Adv.

> **Convention:** During the simulation Sim_C evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by Adv or by Sim_C during its simulation of the corrupted REC. To evaluate $\mathcal{H}(x)$, Sim_C first checks if it has already recorded a pair (x, r), in which case $\mathcal{H}(x)$ evaluates to the value r. Otherwise, Sim_C chooses a random string $r \in \mathbb{Z}_M$, records (x, r) and evaluates $\mathcal{H}(x)$ to r.

1. Upon receiving auxiliary input z, Sim_C invokes Adv on this input.
2. $(\mathsf{preprocess}, n, m)$ messages from the trusted party that denote that the honest SEN submitted a text of length n are ignored.
3. Whenever Adv initializes protocol π_{Query} to learn the trapdoor corresponding to a given pattern, Sim_C invokes the simulator $\mathsf{Sim}_{\mathsf{Query}}$ of the underlying protocol. The simulator $\mathsf{Sim}_{\mathsf{Query}}$ invokes Adv who controls REC in π_{Query}. If $\mathsf{Sim}_{\mathsf{Query}}$ sends $\perp$ to its trusted party, Sim_C also sends $\perp$ to its own trusted party leading to an abort of the execution. Otherwise, Sim_C receives p from $\mathsf{Sim}_{\mathsf{Query}}$ when it sends this value to its own ideal functionality. Finally, Sim_C sends (query, p) to the trusted party.
4. Upon receiving $\perp$ from the ideal functionality, Sim_C sends $\perp$ to Adv and abort. Otherwise, it sends $(\mathsf{approve}, \mathrm{REC})$ to the ideal functionality.
5. Upon receiving $(\mathsf{query}, p, (i_1, \ldots, i_t), \mathsf{id})$ from the trusted party, Sim_C sends $\{i_j\}_{i \in [t]}$ to Adv and outputs whatever Adv does.

Notice that the simulator runs in polynomial time since it runs the polynomial time simulator $\mathsf{Sim}_{\mathsf{Query}}$ for protocol π_{Query}. Next, we show that for a corrupted REC the privacy of sender SEN is guaranteed, and show that REC does not gain any further information during the execution of the real protocol π_{SH}. In fact, the only difference between the simulation above and a real execution of π_{SH} is the way in that the view of REC is generated upon input a query p_i which is approved by SEN. Namely, in the simulation this view is produced by the simulator $\mathsf{Sim}_{\mathsf{Query}}$, while in a real execution it is produced by a run of π_{Query}. Indistinguishability of the two distributions thus follows from the fact that π_{Query} securely implements the oblivious PRF functionality.

Collusion. So far we considered single corruption cases. However, the proof carries over also in the presence of collusion between the receiver REC and S. In this case, we need to show that the server and the receiver cannot conclude any additional information about the text other than what is obtained by the leakage from the queries. A proof of this statement follows the same argument as in the single corruption case. Note in particular that the only additional information given to the server are the values of the patterns, and it still remains infeasible to guess new trapdoors that enable to obtain more information about the outsourced text. ■

4.1 Efficiency

We start by considering the efficiency of our scheme when we do not use the packaging approach. Notice that in this case we do not need the random oracle, but as shown below we have strong limitations on the size of the text. Namely, without packaging the server S is asked to solve subset sum instances of the form $(\widetilde{T}, R_p)$, where $\widetilde{T}$ is a vector of length $\ell = n - m + 1$ with elements from $\mathbb{Z}_M$ for some integer M. To achieve correctness, we require that each subset sum instance has a unique solution with high probability (cf. also the proof of Theorem 1). In order to satisfy this property, one needs to set the parameters in such a way that the quantity $2^\ell/M$ is negligible. Writing $M = 2^{\kappa+\ell}$, we achieve a reasonable correctness level with, e.g., security parameter $\kappa \geq 80$. On the other hand, to let S solve subset sum instances efficiently, we need to consider *low-density* subset sum instances. The analysis of Section 2.2 (see in particular Eq. (2)) yields, in this case, a value of $\ell \approx \sqrt{\kappa}$. This poses an apparently inherent limitation on the length of the text to be preprocessed. For instance, even using a higher value $\kappa \approx 10^4$ (yielding approximately subset sum elements of size 10 KByte) limits the length of the text to only 100 bits.

To overcome this limitations, we can use the packaging approach from our protocol above. Namely, when we structure the text into blocks of length $2m$ bits, the preprocessed blocks $\widetilde{B}_b$ consist of $\ell = m + 1$ elements in $\mathbb{Z}_M$. As above we can set $M = 2^{\kappa+\ell}$ to guarantee correctness. For efficiency, we have, however, the advantage that the blocks are reasonably short which yields subset sum instances of the form $(\widetilde{B}_b, R_p)$ that can be solved in polynomial-time. By combining many blocks we can support texts of *any length* polynomial in the security parameter. We further note that for sufficiently small lengths of m (which are typically some constant), a brute force search in time 2^m per package is sufficient in order to solve the subset sum problem. Finally, we emphasize that the communication/computational complexities of π_{Query} depend on the underlying oblivious PRF evaluation. This in particular only depends on m (due to the complexity of the current implementation of the [NR97] PRF). Using improved PRFs can further reduce the communication complexity. Whereas the communication complexity of π_{Opm} solely depends on the number of matches of p in T which is essentially optimal.

5 Security in the Presence of Malicious Adversaries

In this section we explain how to modify the semi-honest construction from Section 4 to obtain security against malicious adversaries. For simplicity, we will consider progressive modifications of protocol π_{SH}, where each modification deals with a different corruption case. These extensions can then be combined to obtain a construction which supports full malicious security (with the exception that our proof for a malicious sender is not simulation-based). An advantage in such a modular description is the flexibility in picking the identities of the corrupted parties that the system protects against. For instance, in some applications it might be sufficient to protect the system against a corrupted server, while assuming that the clients are semi-honest. In order to maintain the presentation of our protocols simple, we will present them in their basic form without relying on the packaging technique described in Section 4. We stress though that all our constructions can handle packaging as well.

5.1 Dealing with a Malicious Server

The underlying idea here is to add an efficient mechanism in protocol π_{SH} that enables to verify the correctness of the server's answers. Notably, this already provides security against a malicious server, because the server does not have any input/output with respect to protocol π_{Query}, and we already ensured privacy within the semi-honest proof. To do so, we will rely on Merkle trees [Mer89] commitment schemes (see Section 2.3); that essentially produce a succinct commitment that is independent of the length of the com-

mitted message. We modify protocol π_{Pre} by instructing sender SEN to commit to its preprocessed text using Merkle trees. Precisely, let $\widetilde{T}$ be the preprocessed text outsourced to server S, as described in the protocol of Figure 3. Sender SEN generates a binary tree building on top of leaves $\{\widetilde{T}[i]\|i\}_{i=1}^{\ell}$. Then, for every pattern p such that its trapdoor $\mathcal{H}(\mathsf{F}(k,p))$ corresponds to a subset in $\widetilde{T}$ with locations $\{i_1,\ldots,i_t\}$, the server is asked to decommit the paths from the root to the leaves corresponding to these locations. To verify such values, receiver REC recomputes the paths all the way back to the root. We emphasize that this approach already ensures that the server cannot return a proper subset of the actual set of matches, since this would imply that the server has found two different solutions for a given target, which we know occurs only with a negligible probability.

Nevertheless, Merkle trees do not protect against a malicious server declaring that there is no match even though a given pattern actually matched. To prevent this attack we use zero-knowledge sets (see Section 2.6), proving in ZK whether an element is in a set or not. We remark that applying this technique in a naïve way would not work, since the potential number of elements in the set is exponential in n (i.e., counting all possible subsets from the preprocessed text) resulting in exponential running time for SEN and S. Instead, we let SEN commit to the set of trapdoors it generated while creating the preprocessed text $\widetilde{T}$. This yields a much smaller group G with at most ℓ elements. (For privacy issues we need to pad G in such a way that S cannot detect the total number of trapdoors, which would also reveal the number of distinct substrings of length m in the text.) For this set we invoke a zero-knowledge set proof.

Our final construction is slightly different than this, in that we need to provide REC with the proper information allowing it to verify the values retrieved from the server. First, REC needs to know the commitment to $\widetilde{T}$, i.e., the root h of the Merkle tree, and the commitment to the set G (which is denoted h_G). These values will be sent from SEN within protocol π_{Query}. Moreover, in order to verify the ZK set proof, REC also needs the commitment of the trapdoor γ_p. Clearly, this value can neither be forwarded by the server (since otherwise it could easily cheat), nor by sender SEN (since otherwise it should keep a state of size equal to the number of trapdoors). Our solution is to split the output of the PRF into two parts (R'_p, r_p): The first part is used as input to the random oracle $\mathcal{H}$ to evaluate the actual trapdoor R_p; whereas the second part is used as randomness in the computation of the commitment γ_p corresponding to R'_p (and thus to R_p).[7] Given this randomness, REC can compute the commitment γ_p and thus verify the ZK set proof.

To sum up, we rely on the following building blocks: (1) a succinct commitment scheme $(\mathsf{Commit}_{\mathsf{M}}, \mathsf{Open}_{\mathsf{M}})$ (implemented via Merkle trees); (2) a computationally hiding commitment scheme $(\mathsf{Commit}, \mathsf{Open})$; (3) a zero-knowledge sets proof π_{ZKsets}. A detailed description of the protocol can be found in Figure 4.

Theorem 2 *Let $\kappa \in \mathbb{N}$ be the security parameter. For integers n, m, λ, μ, μ' and $\ell = n - m + 1$, let $M = 2^{\ell+\kappa}$, $\mu = \mathsf{poly}(\kappa)$, $\mu' = \mathsf{poly}(\kappa)$, $\lambda = \mathsf{poly}(\kappa)$ and fix κ such that $2^{\ell}/M$ is negligible (in κ). Assume that $\mathcal{H}$ is a random oracle, and that $(\mathsf{Commit}_{\mathsf{M}}, \mathsf{Open}_{\mathsf{M}})$, $(\mathsf{Commit}, \mathsf{Open})$ and π_{ZKsets} are as above. Then, protocol π_{MalS} of Figure 4 securely implements $\mathcal{F}_{\mathrm{OPM}}$ in the presence of a malicious server* S.

Proof: Proving security follows in two steps. We first prove that protocol π_{MalS} is correct. Namely, with all but negligible probability the server does not return a false search response. Second, we claim that condition on that π_{MalS} is correct, we can simulate the view of the server.

Proof of correctness. We say that our construction achieves correctness if with overwhelming probability each pattern query p issued by REC is answered correctly with respect to the outsourced text T. More concretely, for each pattern $p \in \{0,1\}^m$ and a text $T \in \{0,1\}^n$, let $\{i_j\}_{j\in[t]}$ be the positions in T where p

[7] In the basic form of the protocol it does not make any difference whether we commit to R_p or to R'_p. However, in the extension based on packaging committing to R'_p yields a better communication complexity in the setup phase.

Protocol $\pi_{\mathsf{MalS}} = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$

Let $\kappa \in \mathbb{N}$ be the security parameter and let $M, \lambda, m, n, \mu, \mu'$ be integers. Further, let $\mathcal{H} : \{0,1\}^{\mu} \to \mathbb{Z}_M$ be a random oracle and $\mathsf{F} : \{0,1\}^{\kappa} \times \{0,1\}^{m} \to \{0,1\}^{\mu'}$ be a PRF. Consider a succinct commitment scheme $(\mathsf{Commit}_{\mathsf{M}}, \mathsf{Open}_{\mathsf{M}})$, a computationally hiding commitment scheme $(\mathsf{Commit}, \mathsf{Open})$ and a ZK set proof system $(\mathsf{ZKS\text{-}Setup}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$. Protocol π_{MalS} involves a sender SEN holding a text $T \in \{0,1\}^n$, a receiver REC querying patterns $p \in \{0,1\}^m$ and a server S. At the onset, a common reference string $\mathsf{CRS} \leftarrow \mathsf{ZKS\text{-}Setup}(1^{\kappa})$ is generated and distributed to S and REC. The interaction is specified below.

Setup phase, π_{Pre}. The protocol is invoked between sender SEN and server S. (**1**) Preprocessing the text: Given input T and integer m, sender SEN defines a vector $\widetilde{T}$ of length $\ell = n - m + 1$ and picks a random PRF key $k \in \{0,1\}^{\kappa}$. For any $p \in \{0,1\}^m$ denote by $\{i_j\}_{j\in[t]}$ the set of indexes corresponding to the positions where p occurs in T. Sender SEN evaluates $(R'_p, r_p) = \mathsf{F}(k, p)$ (for $R'_p \in \{0,1\}^{\mu}$), computes $R_p = \mathcal{H}(R'_p)$, samples $a_1, \ldots, a_{t-1} \in \mathbb{Z}_M$ at random and then fixes a_t such that $a_t = R_p - \sum_{j=1}^{t-1} a_j \bmod M$. It then sets $\widetilde{T}[i_j] = a_j$ for all $j \in [t]$. (**2**) Committing to $\widetilde{T}$: In addition, SEN commits to $\widetilde{T}$ by letting $h = \mathsf{Commit}_{\mathsf{M}}((\widetilde{T}[1]\|1), \ldots, (\widetilde{T}[\ell]\|(\ell)))$ (for $\{\widetilde{T}[i]\|(i)\}_i$ the committed values). (**3**) Committing to trapdoors: Denote by $p_1, \ldots, p_\nu$ all *distinct* patterns of length m in T, for some $\nu \leq \ell$. Then, SEN computes $\gamma_i = \mathsf{Commit}(R'_{p_i}; r_{p_i})$ for all $i = 1, \ldots, \nu$ and sets $G = \{\gamma_1, \ldots, \gamma_\nu, \gamma_{\nu+1}, \ldots, \gamma_\ell\}$ for randomly chosen $\gamma_{\nu+1}, \ldots, \gamma_\ell$. (**4**) Setup for ZK sets: Finally, SEN computes $(h_G, \sigma) \leftarrow \mathsf{P}_1(1^{\kappa}, \mathsf{CRS}, G)$, outsources $(\widetilde{T}, G, h, \sigma)$ to S and keeps (k, h, h_G).

Query phase, π_{Query}. Upon issuing a query $p \in \{0,1\}^m$ by receiver REC, clients SEN and REC engage in an execution of protocol π_{Query} which implements the oblivious PRF functionality $(k, p) \mapsto (\perp, \mathsf{F}(k, p))$, such that $(R'_p, r_p) = \mathsf{F}(k, p)$. Receiver REC also receives from SEN the commitments h and h_G. Upon completion, REC computes trapdoor $R_p = \mathcal{H}(R'_p)$.

Oblivious pattern matching phase, π_{Opm}. This protocol is engaged between server S that inputs $(\mathsf{CRS}, \widetilde{T}, G, h, \sigma)$ and receiver REC that inputs $(\mathsf{CRS}, h, h_G, R_p, R'_p, r_p)$. Upon receiving (R'_p, γ_p) from REC, where $\gamma_p = \mathsf{Commit}(R'_p; r_p)$, the server views $(\widetilde{T}, R_p = \mathcal{H}(R'_p))$ as a subset sum instance and solves this instance. Let $\mathbf{s}$ denote the solution, and denote with $\{i_j\}_{j\in[t]}$ the set of indexes such that $\mathbf{s}[i_j] = 1$. The server runs $(\widetilde{T}[i_j], \mathrm{path}(i_j)) = \mathsf{Open}_{\mathsf{M}}(h, i_j)$ for all $j \in [t]$ and builds a proof $\pi_{\mathrm{ZKsets}} \leftarrow \mathsf{P}_2(\sigma, \gamma_p)$. Hence, S returns REC the set $\{i_j\}_{j\in[t]}$ together with $\{\widetilde{T}[i_j], \mathrm{path}(i_j)\}_{j\in[t]}$ and π_{ZKsets}. Receiver REC verifies the openings $\{\widetilde{T}[i_j], \mathrm{path}(i_j)\}_{j\in[t]}$, checks that $\sum_{j=1}^{t} \widetilde{T}[i_j] = R_p$ and that $\mathsf{V}(1^{\kappa}, \mathsf{CRS}, h_G, \gamma_p, \pi_{\mathrm{ZKsets}}) = 1$.

Figure 4: Outsourced pattern matching resisting a malicious server

matches the text. Then, our protocol is correct if it returns for any such p and T the same matches $\{i_j\}_{j\in[t]}$. As in Theorem 1, we neglect the event that a solution of subset sum is non-unique due to a collision in the PRF and/or random oracle (since the probability that this happens is negligible in the security parameter κ).

For each pattern p we consider two subcases here. In case p matches the text in at least one position, we reduce correctness to the collision intractability of Merkle trees and the soundness of the ZK set proof. In case p does not match the text, we again rely on the collision intractability of Merkle trees.

Claim 4 *For any pattern $p^* \in \{0,1\}^m$ such that p^* matches T in at least one position, protocol π_{Query} is correct assuming the binding property of Merkle trees and the negligible soundness of π_{ZKsets}.*

Proof: Consider a pattern p^* and assume that it matches the text in positions $\tau = \{i_j\}_{j\in[t]}$. We consider two bad events and bound their probability.

- bad_1: Occurs in the real execution whenever S convinces REC that p^* does not match the text.

- bad_2: Occurs in the real execution whenever S convinces REC that the solution to the subset sum instance $(\widetilde{T}, R_{p^*})$ is $\tau' = \{i'_j\}_{j' \in [t']}$ for $\{i'_j\}_j \neq \{i_j\}_j$.

Intuitively, bad_1 happens with negligible probability since provoking this event means that the server is able to prove that the commitment to a trapdoor R_p does not belong to the set G of all commitments. This violates the soundness property of π_{ZKsets}.

We start by considering a malicious Adv controlling S in the real world and provoking event bad_1 for text T and pattern p^*. From such an adversary, we build an efficient machine P^* breaking soundness of $(\mathsf{ZKS\text{-}Setup}, (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$, namely P^* is given as input $\mathsf{CRS} \leftarrow \mathsf{ZKS\text{-}Setup}(1^\kappa)$ and is able to find a set G and a tuple $(h_G, \pi'_{\text{ZKsets}}, \pi''_{\text{ZKsets}})$ for a statement $\gamma^* \in G$, such that $\mathsf{V}(1^\kappa, \mathsf{CRS}, \gamma^*, h_G, \pi'_{\text{ZKsets}})$ outputs 0 and $\mathsf{V}(1^\kappa, \mathsf{CRS}, \gamma^*, h_G, \pi''_{\text{ZKsets}})$ outputs 1. Prover P^* proceeds as follows. It first processes the text T as described in the protocol of Figure 4, obtaining $(\widetilde{T}, G, h, h_G, \sigma)$ such that $(\sigma, h_G) \leftarrow \mathsf{P}_1(1^\kappa, \mathsf{CRS}, G)$. The values $(\widetilde{T}, G, h, \sigma)$ are forwarded to S together with the common reference string CRS. Furthermore, P^* implicitly sets $\gamma^* = \gamma_{p^*}$ and computes $\pi'_{\text{ZKsets}} = \mathsf{P}_2(\sigma, \gamma^*)$. At this point P^* simulates a query for pattern p^* (matching T by hypothesis), as described in protocol π_{MalS}; this yields a pair (R'_{p^*}, r_{p^*}) which is forwarded to the server. S returns the matching positions $\{i_j\}_{j \in [t]}$ together with a proof π''_{ZKsets}.

Note that the above simulation is perfect from the point of view of S. Hence, P^* outputs $(h_G, \gamma_{p^*}, \pi'_{\text{ZKsets}}, \pi''_{\text{ZKsets}})$. This contradicts soundness, since π'_{ZKsets} will not accept (as it was generated honestly and p^* matches), whereas π''_{ZKsets} will accepts (as the above is a perfect simulation and bad_1 happens).

Next, consider the second event bad_2. Intuitively, bad_2 happens with a negligible probability since provoking this event means that the server is able to decommit at least one position in the text to a non-consistent value. This violates the binding property of Merkle trees. Recall that, with overwhelming probability, there exists at most one solution to the subset sum instance $(\widetilde{T}, R_{p^*})$. Denote this "non-collision event" by nocol. We can write:

$$\Pr[\mathsf{bad}_2] = \Pr[\mathsf{bad}_2|\mathsf{nocol}] \cdot \Pr[\mathsf{nocol}] + \Pr[\mathsf{bad}_2|\overline{\mathsf{nocol}}] \cdot \Pr[\overline{\mathsf{nocol}}] \leq \Pr[\mathsf{bad}_2|\mathsf{nocol}] + \Pr[\overline{\mathsf{nocol}}].$$

Due to the low collision probability, we have $\Pr(\overline{\mathsf{nocol}}) \leq \mathsf{negl}(\kappa)$. It remains to prove that $\Pr(\mathsf{bad}_2|\mathsf{nocol}) \leq \mathsf{negl}(\kappa)$. However, conditioned on nocol, the only way to provoke event bad_2 is by violating the binding property of Merkle trees: There exists an index $i_j \in \tau'$ such that $\mathsf{Open}_{\mathsf{M}}(h, i_j) \neq \widetilde{T}[i'_j]$. In particular, an adversary provoking this event can be turned into a concrete polynomial-time machine breaking collision resistance of $(\mathsf{Commit}_{\mathsf{M}}, \mathsf{Open}_{\mathsf{M}})$. Thus, $\Pr[\mathsf{bad}_2|\mathsf{nocol}]$ must also be negligible. Together with the previous equation, this concludes the proof of the claim. ■

Claim 5 *For any pattern $p^* \in \{0,1\}^m$ such that p^* does not match T, protocol π_{Query} is correct assuming the binding property of Merkle trees and the negligible soundness of π_{ZKsets}.*

This proof follows similarly to the proof of Claim 4 and is therefore omitted.

Privacy. Finally, we note that conditioned on correctness, simulating the server's view reduces almost immediately to the semi-honest case; so essentially the same simulator as in the proof of Theorem 1 will do. We only need to specify how the simulator can simulate the additional messages which are included in protocol π_{MalS} but not in π_{SH}. Specifically, during the setup phase Sim needs also to produce the values (G, h, σ). To do so, the simulator will simply commit to ℓ random values and then compute σ as a function of G by running P_1 as in a real execution. On the other hand, the value h can be computed consistently with the sampled pre-processed text $\widetilde{T}$. Finally, during the oblivious pattern matching phase, Sim will first

compute the value R_p as described in the proof of Theorem 1. If the pattern matches, then γ_p is chosen to be a random element in G; otherwise a fresh commitment of a random message is used. It follows from the computationally hiding property of $(\mathsf{Commit}, \mathsf{Open})$ that the above strategy is not detectable by the adversary but with a negligible probability.

A formal description of the simulator $\mathsf{Sim}_{\mathrm{S}}$ (with access to Adv) follows.

Convention: During the simulation $\mathsf{Sim}_{\mathrm{S}}$ evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by Adv or by $\mathsf{Sim}_{\mathrm{S}}$ during its simulation of the corrupted server. To evaluate $\mathcal{H}(x)$, $\mathsf{Sim}_{\mathrm{S}}$ first checks if it has already recorded a pair (x, r), in which case $\mathcal{H}(x)$ evaluates to the value r. Otherwise, $\mathsf{Sim}_{\mathrm{S}}$ chooses a random string $r \in \mathbb{Z}_M$, records (x, r) and evaluates $\mathcal{H}(x)$ to r.

1. On input the auxiliary input z, $\mathsf{Sim}_{\mathrm{S}}$ invokes Adv (i.e., the corrupted server) on this input. The simulator keeps track of a table $\mathcal{B}$ that is initially set to the empty table.

2. Upon receiving a $(\mathsf{preprocess}, n, m)$ message from the trusted party denoting that the honest SEN wants to outsource a text of length n to the trusted party, $\mathsf{Sim}_{\mathrm{S}}$ defines text $\widetilde{T}$ by sampling uniformly at random a vector of length $\ell := n - m + 1$ from $\mathbb{Z}_M^{\ell}$.

   Next, $\mathsf{Sim}_{\mathrm{S}}$ computes $h = \mathsf{Commit}_{\mathsf{M}}((\widetilde{T}[1] \| (1)), \ldots, (\widetilde{T}[\ell] \| (\ell)))$ (for $\{\widetilde{T}[i] \| (i)\}_i$ the committed values), lets $G = \{\gamma_1, \ldots, \gamma_\ell\}$ for randomly chosen $\gamma_1, \ldots, \gamma_\ell$, and sets $(h_G, \sigma) \leftarrow \mathsf{P}_1(1^\kappa, \mathsf{CRS}, G)$. Hence, it forwards $(\widetilde{T}, G, h, \sigma)$ to adversary Adv and stores $(\widetilde{T}, G)$ for later usage as well.

3. Upon receiving a $(\mathsf{query}, \mathrm{REC}, (i_1, \ldots, i_t), \mathsf{id})$ message from the trusted party indicating that receiver REC submitted a search query that was approved by SEN, $\mathsf{Sim}_{\mathrm{S}}$ distinguishes two cases.

   (a) *Pattern queried by* REC *appears in* T*:* In this case $\{i_j\}_{j \in [t]}$ is not the empty set, and the simulator samples uniformly at random a value X_{id} from $\{0,1\}^\mu$. Then, it programs the random oracle $\mathcal{H}(X_{\mathsf{id}})$ to $\sum_{j=1}^{t} \widetilde{T}[i_j]$; if $\mathcal{H}$ has already been programmed to a different value, then we abort. Furthermore, $\mathsf{Sim}_{\mathrm{S}}$ picks an element $\gamma \leftarrow G$.

   (b) *Pattern does not appear in* T*:* In this case $\{i_j\}_{j \in [t]}$ is the empty set, and we check if table $\mathcal{B}$ contains a value of the form $(\mathsf{id}, X_{\mathsf{id}})$. Otherwise, we pick the value X_{id} uniformly at random in $\{0,1\}^\mu$ and store $(\mathsf{id}, X_{\mathsf{id}})$ in $\mathcal{B}$. Furthermore, $\mathsf{Sim}_{\mathrm{S}}$ picks an element γ by committing to a random message.

   Finally, $\mathsf{Sim}_{\mathrm{S}}$ (emulating the role of REC in the real execution) forwards $(X_{\mathsf{id}}, \gamma)$ to the adversary.

4. If Adv does not answer with $(i_1, \ldots, i_t)$, $\mathsf{Sim}_{\mathrm{S}}$ sends $\perp$ to ideal functionality and abort. Otherwise it sends the trusted party $(\mathsf{approve}, \mathrm{REC})$.

5. $\mathsf{Sim}_{\mathrm{S}}$ outputs whatever Adv does.

We first note that $\mathsf{Sim}_{\mathrm{S}}$ runs in polynomial time since it only samples a random vector from $\mathbb{Z}_M^{\ell}$, and then calculates the sum of values from a given subset. Next, we show that the distribution produced by $\mathsf{Sim}_{\mathrm{S}}$ in the ideal world is computationally indistinguishable from the distribution that Adv expects to see in the real world. This is required to hold even given the leakage revealing the positions where the pattern matches the text. We start by defining a hybrid distribution $\mathbf{HYB}^1_{\pi_{\mathsf{MalS}}, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ that is defined as the real experiment $\mathbf{REAL}_{\pi_{\mathsf{MalS}}, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ with the difference that X_{id} is not computed by a PRF but rather by a random function f_κ. We have the following claim:

Claim 6 *Let* F *be a secure pseudorandom function (cf. Definition 1), then for all sufficiently large* $\kappa \in \mathbb{N}$ *and for any tuple of inputs* $(T, (p_1, \ldots, p_\lambda))$ *and auxiliary input* z*, it holds that*

$$\{\mathbf{REAL}_{\pi_{\mathsf{MalS}}, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa \in \mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{HYB}^1_{\pi_{\mathsf{MalS}}, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa \in \mathbb{N}}.$$

The proof of the above claim is similar to the proof of Claim 2, and consists of a simple reduction to the pseudorandomness property of the PRF.

Next we consider a hybrid distribution $\mathbf{HYB}^2_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ that is defined as the previous hybrid, with the difference that the elements in the set G are replaced by commitments to random messages. A standard hybrid argument yields the following claim:

Claim 7 *Let* $(\mathsf{Commit},\mathsf{Open})$ *be computationally hiding, then for all sufficiently large* $\kappa\in\mathbb{N}$ *and for any tuple of inputs* $(T,(p_1,\ldots,p_\lambda))$ *and auxiliary input* z*, it holds that*

$$\{\mathbf{HYB}^1_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}\stackrel{\mathrm{c}}{\approx}\{\mathbf{HYB}^2_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}.$$

Finally, we need to bound the distance between the experiment $\mathbf{HYB}^2_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ and the simulated view. We do this in the next claim.

Claim 8 *For any input text* T*, patterns* $p_1,\ldots,p_\lambda$*, and auxiliary input* z*, it holds that*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}\equiv_s\{\mathbf{HYB}^2_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))\}_{\kappa\in\mathbb{N}}.$$

Proof: Define the following event bad_3 that occurs when the simulator aborts in the ideal world.

- Event bad_3: Occurs if the simulation given above is aborted. In this case the corrupted server has asked for a direct query to the random oracle of the form $(f_\kappa(p)|_\mu)$ (where $y|_\mu$ denotes the truncation of bit-string y to the first μ bits) *before* it has seen $f_\kappa(p)$, where p is a pattern that occurs in the text T. Recall that f_κ is a random function as defined in the first hybrid world.

An argument similar to the proof of Theorem 1 shows that the simulated view and the output distribution in $\mathbf{HYB}^2_{\pi_{\mathsf{MalS}},\mathsf{Adv}(z)}(\kappa,(-,T,(p_1,\ldots,p_\lambda)))$ are identical conditioned on the event $\mathsf{bad}:=\mathsf{bad}_1\wedge\mathsf{bad}_2\wedge\mathsf{bad}_3$ not happening.

On the other hand, since f_κ is a random function, it is easy to see that bad_3 occurs with probability at most $\mathsf{poly}(\kappa)/2^\mu=\mathsf{negl}(\kappa)$. Now, Claim 4 and Claim 5 together with a union bound imply that the probability of bad is negligible. This concludes the proof. ■

Putting together Claim 6, Claim 7 and Claim 8 concludes the proof. ■

5.2 Dealing with a Malicious Receiver

We note that the protocol of Figure 3 is already secure against a malicious REC (with a small change). In fact, the only inputs receiver REC provides in an execution of protocol π_{SH} (sub-protocol π_{Query}) are the search queries p_i's. To this end, we claim that the only way REC can attack the protocol is by guessing a trapdoor of which it did not submit a query for. We further claim that the probability of this event is negligible in the security parameter. Loosely speaking, this follows from the security of the PRF. Namely, given that this event occurs with a non-negligible probability a distinguisher can detect this trapdoor in the queries submitted to the random oracle. The reduction to the hardness of the PRF follows the same outlines as the semi-honest reduction for the case that the server is corrupted and is therefore omitted. Hence, it suffices to ensure that protocol π_{Query} is secure in the presence of malicious adversaries so that a simulator can extract these queries. (An example is the protocol of [HL10], taken from [FIPR05], which implements the Naor-Riengold function [NR97] in the presence of malicious adversaries.)

5.3 Dealing with a Malicious Sender

The most difficult case to deal with, is proving security against a corrupted SEN. The reasons for this are as follows. First, we must ensure that the subset sum vector $\widetilde{T}$ corresponds to a well defined text T, so that the simulator will be able to extract it. In systems that rely on a random oracle, this issue is typically addressed using the cut-and-choose technique which is rather inefficient; see [IKNP03] for one example. Another difficulty is that we need to ensure that sender SEN uses consistent trapdoors when running against REC and S. In fact, SEN could potentially use two different trapdoors in these executions or use a maliciously chosen key (that leads to a collision for two different patterns), causing the simulation to fail easily. In this section we design a protocol that is comprised of a sequence of steps which are backed-up with efficient zero-knowledge (ZK) proofs, so that verification of the pre-processed text is run "on-the-fly" during the query phase. Namely, the server is handed a trapdoor for derandomizing the computations of SEN with respect to each query. It then recomputes whatever SEN has computed in the setup phase and verifies the correctness of these computations. Nevertheless, we cannot construct a simulation-based security proof for this protocol since it is not clear how to design a ZK proof for computations that are based on a random oracle. Instead, we design a new protocol and prove that it maintains privacy and correctness. We continue with the description of the modifications needed for protocol π_{SH} from Figure 3. We first give a high level overview, a more formal description can be found in Figure 5. More concretely, our protocol includes the following steps:

- *Commitment to the text.* The sender SEN commits to all values $T[i]$ for $i = 1, \ldots, n$. The resulting commitments c_i are forwarded to S together with a proof that the committed value is a bit. This can be done efficiently using the ZK proof system π_{isBit} (cf. Appendix A.1).

- *Commitment to the PRF key.* In the next step, SEN samples a PRF key k and commits to it. At this point the server S picks a fresh random key k' and sends it to SEN. The parties then compute the commitment of $k \cdot k'$. This step requires using a homomorphic commitment.

- *Derandomization.* In order to derandomize SEN's computations, SEN samples another PRF key k'' and commits to it. Let $T_i^m = (T[i], \ldots, T[i+m-1])$ be the ith substring of length m in T for $i \in [n-m+1]$. Then for every substring T_i^m the sender feeds the pseudo-random generator PRG with input $\mathsf{F}(k'', T_i^m)$; denote with $(r_{\mathsf{F}}^i, r_{\text{isPRF}}^i, r_{\text{isBit}}^i)$ the output of the PRG. The sender will use this randomness for all its computations from here on.

- *Committed PRF evaluations.* SEN sends S a vector of commitments $\mathbf{c}' = (c'_1, \ldots, c'_{n-m+1})$ that specifies $c'_i = \mathsf{Commit}(\mathsf{F}(k \cdot k', T_i^m; r_{\mathsf{F}}^i))$. Next, SEN and S engage in an execution of protocol π_{CPRF} that securely implements $\mathcal{F}_{\text{CPRF}}$ in the presence of malicious adversaries, such that SEN enters key $k \cdot k'$ and text T, whereas S enters the commitment to key $k \cdot k'$, the commitments to $\{T_i^m\}_i$ and $\mathbf{c}'$. This proof ensures that SEN computes the PRF evaluations correctly with respect to $k \cdot k'$; see Appendix A.2 for the complete details of π_{CPRF}. Let b denote the server's outcome, then the server continues to the next step only if $b = 1$.

- *Commitment to the number of matches.* Let $\mathbf{c}'' = (c''_1, \ldots, c''_{n-m+1})$ be the permuted version of $\mathbf{c}'$ such that the elements are sorted in a non-decreasing order with respect to the corresponding plaintexts T_i^m. Now, SEN forwards $\mathbf{c}''$ to S together with a proof that the sorting is done correctly. Such a proof can be implemented efficiently using the proof system π_{isSort} (cf. Appendix A.2).

  For each $i \in [n-m]$ denote with $\tilde{\mathbf{c}}$ the vector containing $\tilde{c}_i = c''_{i+1}/c''_i$. It follows from the homomorphic property of the commitment scheme that $\tilde{c}_i$ commits to 0 whenever c''_{i+1} and c''_i commits the same value. SEN provides an "indicator vector" $\hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_{n-m})$, such that $\hat{c}_i$ commits to zero

if and only if $\tilde{c}_i$ commits to a plaintext *different* than 0 and otherwise it commits to 1. Hence, SEN first proves that $\hat{\mathbf{c}}$ commits to bits using π_{isBit}. In addition, for each i, it proves that the product of the plaintexts underlying $\tilde{c}_i$ and $\hat{c}_i$ equals 0. This implies that whenever $\tilde{c}_i \in \tilde{\mathbf{c}}$ commits to a non-zero value, the corresponding commitment $\hat{c}_i \in \hat{\mathbf{c}}$ commits to zero. Finally, SEN proves that $\tilde{c}_i \cdot \hat{c}_i$ commits to a non-zero value. This implies that whenever $\tilde{c}_i \in \tilde{\mathbf{c}}$ commits to 0, the corresponding commitment $\hat{c}_i \in \hat{\mathbf{c}}$ commits to 1. Note that in order to verify the number of matches of a query p, it is sufficient to check the number of consecutive 1's in $\hat{\mathbf{c}}$ in the positions associated with p, see details below. The last two proofs can be carried out using the ZK proof systems π_{mult} and π_{NonZero} (cf. Appendix A.1). Notably, the commitments in $\mathbf{c}''$ and $\hat{\mathbf{c}}$ are computed using randomness r^i_{isSort} and r^i_{isBit}.

We now describe the modified protocol π_{Query}, between SEN and REC. Here, the parties run a protocol that implements the functionality $\mathcal{F}_{\text{2PRF}} : \big((k_0, k_1), (p, \mathsf{Commit}(k_0), \mathsf{Commit}(k_1))\big) \mapsto \big(-, \mathsf{F}(k_0, p), \mathsf{F}(k_1, p)\big)$. Let π_{2PRF} denote a protocol that implements $\mathcal{F}_{\text{2PRF}}$ in the presence of malicious adversaries. An implementation of this functionality based on the [NR97] PRF can be found in Appendix A.2.

It remains to modify protocol π_{Opm} between receiver REC and server S. First the two parties verify that their commitments for the PRF keys are identical. Next, REC sends the values $(\mathsf{F}(k \cdot k', p), \mathsf{F}(k'', p))$ to S. The first value is used to solve the corresponding subset sum instance as in our protocol π_{SH}. The second value is used as input to the pseudo-random generator PRG to extract the randomness used by SEN in computing the commitments associated with pattern p and verify the computations performed by SEN in the setup phase. If these checks fail then the server returns an abort message. Otherwise it returns the matched text positions. Next, we prove the following result.

Theorem 3 *Let $\kappa \in \mathbb{N}$ be the security parameter. For integers n, m, λ, μ and $\ell = n - m + 1$, let $M = 2^{\ell+\kappa}$, $\mu = \mathsf{poly}(\kappa)$, $\lambda = \mathsf{poly}(\kappa)$ and fix κ such that $2^\ell / M$ is negligible (in κ). Assume that $\mathcal{H}$ is a random oracle, that all the ZK proofs are sound, that $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ is a homomorphic commitment scheme and that $\mathcal{F}_{\text{CPRF}}$ and $\mathcal{F}_{\text{2PRF}}$ are implemented using protocols that are secure in the presence of malicious adversaries. Then, protocol π_{MalSEN} of Figure 5 privately and correctly computes outsourced pattern matching in the presence of a malicious sender* SEN.

Proof: We first argue about privacy and then correctness.

Privacy. We prove privacy first. The privacy argument implies that for any two sets of queries $(p_1, \ldots, p_\lambda)$ and $(p'_1, \ldots, p'_\lambda)$ of the same length, the sender cannot distinguish between a (sequential) execution against the receiver with queries $(p_1, \ldots, p_\lambda)$ and a (sequential) execution with queries $(p'_1, \ldots, p'_\lambda)$. This argument follows similarly to the argument made in the proof of protocol π_{SH}. Namely, the sender cannot learn anything about the query since the protocol for which the sender and the receiver engage with is secure in the presence of malicious attacks. More formally, denote the sender's view within π_{Query} by $\mathbf{VIEW}_{\pi_{\mathsf{Query}}, \text{SEN}(z)}(\kappa, (-, T, p_1, \ldots, p_\lambda))$. Then,

$$\mathbf{VIEW}_{\pi_{\mathsf{Query}}, \text{SEN}(z)}(\kappa, (-, T, p_1, \ldots, p_\lambda)) \stackrel{\text{c}}{\approx} \mathbf{VIEW}_{\pi_{\mathsf{Query}}, \text{SEN}(z)}(\kappa, (-, T, p'_1, \ldots, p'_\lambda)).$$

Indistinguishability boils down to $\lambda - 1$ hybrid arguments for which for every $j \in [\lambda]$

$$\begin{aligned} &\mathbf{VIEW}^i_{\pi_{\mathsf{Query}}, \text{SEN}(z)}(\kappa, (-, T, p_1, \ldots, p_i, p'_{i+1}, \ldots p_\lambda)) \\ &\stackrel{\text{c}}{\approx} \quad \mathbf{VIEW}^{i+1}_{\pi_{\mathsf{Query}}, \text{SEN}(z)}(\kappa, (-, T, p_1, \ldots, p_{i+1}, p'_{i+2}, \ldots p_\lambda)). \end{aligned}$$

Indistinguishability holds due to the security of π_{Query}. Namely, assume that for some fixed $j \in [\lambda]$ there exists a distinguisher D for which views $\mathbf{VIEW}^j$ and $\mathbf{VIEW}^{j+1}$ are distinguishable with a non-negligible

Protocol $\pi_{\mathsf{MalSEN}} = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$

Let $\kappa \in \mathbb{N}$ be the security parameter and let M, λ, m, n, μ be integers. Further, let $\mathcal{H} : \{0,1\}^{\mu} \to \mathbb{Z}_M$ be a random oracle, $\mathsf{F} : \{0,1\}^{m+1} \times \{0,1\}^m \to \{0,1\}^{\mu}$ be a pseudo-random function, $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open})$ be a homomorphic commitment scheme and $\mathsf{PRG} : \{0,1\}^{\mu} \to \{0,1\}^*$ be a pseudo-random generator. Protocol π_{MalSEN} involves a sender SEN holding a text $T \in \{0,1\}^n$, a receiver REC querying for patterns $p \in \{0,1\}^m$, and a server S. The interaction is specified below.

Setup phase, π_{Pre}**.** The protocol is invoked between sender SEN and server S and is introduced in a sequence of phases.

- *Input preprocessing.* SEN executes the same preprocessing phase from π_{SH}, yielding a vector $\widetilde{T}$. Then, the sender computes $c_i \leftarrow \mathsf{Commit}(T[i])$ for all $i \in [n]$ and forwards $\{c_i, \pi_{\text{isBit}}\}_{i\in[n]}$ to S.
- *Committing to PRF evaluations.* SEN picks random keys k and k'' and releases their commitments $c_{\mathsf{F}} \leftarrow \mathsf{Commit}(k)$ and $c''_{\mathsf{F}} \leftarrow \mathsf{Commit}(k'')$. S picks a random key k' and sends it to SEN. The parties compute $c'_{\mathsf{F}} = \mathsf{Commit}(k \cdot k')$. Let $T_i^m = (T[i], \ldots, T[i+m-1])$ be the ith substring of length m in T. Then, SEN computes $(r^i_{\mathsf{F}}, r^i_{\text{isSort}}, r^i_{\text{isBit}}) = \mathsf{PRG}(\mathsf{F}(k'', T_i^m))$ and sends $\{c'_i = \mathsf{Commit}(\mathsf{F}(k \cdot k', T_i^m); r^i_{\mathsf{F}}), \pi_{\text{isPRF}}\}_{i\in[n-m+1]}$ (where each commitment is computed with independent randomness).

  Next, SEN and S invoke π_{CPRF} such that SEN enters $k \cdot k'$ and T, whereas S enters c'_{F}, the commitments to $\{T_i^m\}_i$ and $\mathbf{c}'$. S aborts if its outcome from π_{CPRF} is zero.
- *Committing to number of matches.* SEN sends $\{c''_i, \pi_{\text{isSort}}\}_{i\in[n-m+1]}$, computes $\tilde{c}_i = c''_{i+1}/c''_i$ and forwards the indicator vector $\{\hat{c}_i, \pi_{\text{isBit}}\}_{i\in[n-m]}$. Then, for each i, sender SEN sends $(\tilde{c}_i, \hat{c}_i, \mathsf{Commit}(0), \pi_{\text{mult}})$ and $(\tilde{c}_i \cdot \hat{c}_i, \pi_{\text{NonZero}})$ where the commitments c''_i and $\hat{c}_i$ are computed using randomness r^i_{isSort} and r^i_{isBit} respectively.

The server verifies these proofs and outputs $\perp$ whenever the verification fails.

Query phase, π_{Query}**.** Upon issuing a query $p \in \{0,1\}^m$ by receiver REC, clients SEN and REC engage in an execution of protocol π_{2PRF}. Denote REC's outputs by (R_p, R'_p), respectively. SEN further sends the values c'_{F} and c''_{F}.

Oblivious pattern matching phase, π_{Opm}**.** This protocol is engaged between server S and receiver REC which first check for equality of the commitments of $k \cdot k'$ and k''. In case equality holds, REC forwards the server the values (R_p, R'_p). The server views $(R_p, \widetilde{T})$ as a subset sum instance and solves this instance. Let $\mathbf{s}$ be the solution of this instance and denote with $\{i_j\}_{j\in[t]}$ the set of indexes such that $\mathbf{s}[i_j] = 1$ (if such exist). At this point, S computes $\mathsf{PRG}(R'_p)$ and checks SEN's computations with respect to p as follows. It first computes $\{\mathsf{Commit}(R_p; r^j_{\mathsf{F}})\}_{j\in[t]}$ and compares the outcome to the commitments in $\mathbf{c}'$ within positions $\{i_j\}_{j\in[t]}$. In addition, S computes $\{\mathsf{Commit}(R_p; r^j_{\text{isSort}})\}_{j\in[t]}$ and checks for t consecutive commitments within $\mathbf{c}''$ that equal exactly the obtained values. Lastly, S computes $\{\mathsf{Commit}(0; r^1_{\text{isBit}}), \{\mathsf{Commit}(1; r^j_{\text{isBit}})\}_{j\in[t]\setminus 1}, \mathsf{Commit}(0; r^{t+1}_{\text{isBit}})\}$ (where the first/last commitment of 0 is omitted if necessary) and checks for $t+1$ consecutive commitments $\{\hat{c}_{i^*_j}\}_{j\in[t+1]}$ within $\hat{\mathbf{c}}$ that equal the obtained values, where $i_j - 1 = i^*_j$ for $j \in [t]$ and $i_t = i^*_{t+1}$. If all verifications follow correctly, S returns REC the set $\{i_j\}_{j\in[t]}$. Otherwise, it returns an abort message.

Figure 5: Outsourced pattern matching resisting a malicious sender SEN

probability. We construct a new distinguisher D_{Query} that interacts with an external receiver and an internal malicious sender, that distinguishes a single execution of π_{Query} and is defined as follows. Given index j, D_{Query} emulates the role of the honest receiver in the first j executions of π_{Query} using queries $p_1, \ldots, p_j$.

In the $j+1$ execution, D_{Query} interacts with the external receiver and forwards its messages to the malicious sender. Finally, D_{Query} emulates the remaining executions of π_{Query}, playing the role of the honest receiver with input $p'_{j+2}, \ldots, p'_{\lambda}$.

Correctness. The correctness argument is more subtle. Here we need to make sure that the receiver learns the correct matched text locations with respect to T and $\widetilde{T}$. Specifically, we prove that with all but negligible probability, the receiver learns the correct matched text locations in T relative to its queries. For each query p the receiver learns two trapdoors $\mathsf{F}(k \cdot k', p)$ and $\mathsf{F}(k'', p)$. Recall that the first trapdoor is used for solving a subset sum instance whereas the second trapdoor is used for verifying the computations of the sender. Now, we claim first that any matched text location that is returned by the subset sum solution is a valid position in T for which the pattern matches the text. This is because the server is able to precisely verify the computations of the sender. That is, the server extracts the randomness used to compute the commitments to the PRF evaluations and is able to recompute these commitments and check that their positions in the vector of commitments match the positions returned by the subset sum solution. On the other hand, any matched position in T will be a matched position in $\widetilde{T}$ with overwhelming probability, otherwise the server will aborts. This is because the text is being verified against the PRF evaluations, where these values are being sorted. Therefore, the sender cannot report less matched positions than the actual number. ■

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[BM77] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[CFG89] Mark Chaimovich, Gregory Freiman, and Zvi Galil. Solving dense subset-sum problems by using analytical number theory. *J. Complexity*, 5(3):271–282, 1989.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, 1997.

[CHL+05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *EUROCRYPT*, pages 422–439, 2005.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[CRFM11] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. *IEEE Transactions on Information Theory*, 57(4):2488–2502, 2011.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, pages 125–142, 2002.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[DMV13] Özgür Dagdelen, Payman Mohassel, and Daniele Venturi. Rate-limited secure function evaluation: Definitions and constructions. In *Public Key Cryptography*, pages 461–478, 2013.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[FP05] Abraham Flaxman and Bartosz Przydatek. Solving medium-density subset sum problems in expected polynomial time. In *STACS*, pages 305–314, 2005.

[Fri86] Alan M. Frieze. On the lagarias-odlyzko algorithm for the subset sum problem. *SIAM J. Comput.*, 15(2):536–539, 1986.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GHS10] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Text search protocols with simulation based security. In *Public Key Cryptography*, pages 332–350, 2010.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, 1990.

[GM91] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. In *ICALP*, pages 719–727, 1991.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptology*, 23(3):422–456, 2010.

[HN12] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3):383–433, 2012.

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.

[HZ14] Carmit Hazay and Hila Zarosim. The feasibility of outsourced database search in the plain model. In *Manuscript*, 2014.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, pages 97–114, 2007.

[KM10] Jonathan Katz and Lior Malka. Secure text processing with applications to private dna matching. In *ACM Conference on Computer and Communications Security*, pages 485–492, 2010.

[KMP77] Donald E. Knuth, James H. Jr. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *ACM Conference on Computer and Communications Security*, pages 797–808, 2012.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[Lip03] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In *ASIACRYPT*, pages 398–415, 2003.

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, pages 382–400, 2010.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *APPROX-RANDOM*, pages 378–389, 2005.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, pages 218–238, 1989.

[Moh11] Payman Mohassel. Efficient and secure delegation of linear algebra. *IACR Cryptology ePrint Archive*, 2011:605, 2011.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *FOCS*, pages 80–91, 2003.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467, 1997.

[NS06] Phong Q. Nguyen and Damien Stehlé. LLL on the average. In *ANTS*, pages 238–256, 2006.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[PTT11] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *CRYPTO*, pages 91–110, 2011.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[Sha08] Andrew Shallue. An improved multi-set algorithm for the dense subset sum problem. In *ANTS*, pages 416–429, 2008.

[TPKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient dna searching through oblivious automata. In *ACM Conference on Computer and Communications Security*, pages 519–528, 2007.

A Building Blocks

In the following section we overview zero-knowledge (ZK) proofs that are important for our construction in the malicious setting to ensure correctness. All of these proofs are Σ-protocols over a group $\mathbb{G}$ of composite order, and adapted from proofs for prime order groups. In Section A.2 we describe additional sub-protocols that include more involved zero-knowledge proofs and protocols for correctness.

A.1 Zero-Knowledge Proofs with Constant Overhead

1. Proving the knowledge of a discrete logarithm of a statement x [Sch89].

$$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}$$

   Denote this proof by π_{DL}.

2. Proving that a commitment c either commits to 0 or 1. For our particular instantiation of commitment scheme, this can be obtained directly from π_{DL} using the compound proof of Cramer et al. [CGS97].

$$\mathcal{R}_{\text{isBit}} = \{((\mathbb{G}, \mathsf{pk}, p, c), (b, r)) \mid c = \mathsf{Commit}(\mathsf{pk}, b; r) \wedge b \in \{0, 1\}\}.$$

   Denote this proof by π_{isBit}.

3. Proving that a commitment c_2 commits to the product of two decommitted values [DF02, DJN10]. Namely,

$$\mathcal{R}_{\text{mult}} = \left\{ ((\mathbb{G}, \mathsf{pk}, c_0, c_1, c_2), (a_0, a_1, r_0, r_1, r_2,)) \text{ s.t. } \begin{array}{l} c_i = \mathsf{Commit}(\mathsf{pk}, a_i; r_i) \text{ for } i \in \{0, 1\} \wedge \\ c_2 = \mathsf{Commit}(\mathsf{pk}, a_0 \cdot a_1; r_2) \end{array} \right\}$$

   Denote this proof by π_{mult}.

4. Proving that a commitment c commits to a non-zero value [HN12]. Namely,

$$\mathcal{R}_{\text{NonZero}} = \{(\mathbb{G}, \mathsf{pk}, c) \mid \exists\, (x \neq 0, r) \text{ s.t. } \mathsf{Commit}(\mathsf{pk}, x; r)\}.$$

   Denote this proof by π_{NonZero}.

A.2 Additional Tools

A.2.1 Committed Sorted Set

In this section we consider a proof for proving that a set of decommitted values is sorted. Namely,

$$\mathcal{R}_{\text{isSort}} = \left\{ \left(\mathsf{pk}, \{c_i\}_{i \in [\tau]}\right), \left(\{p_i, r_i\}_{\in [\tau]}\right) \middle| \forall\, i,\ c_i = \mathsf{Commit}(\mathsf{pk}, p_i; r_i) \wedge p_{i+1} \geq p_i \right\}.$$

This proof can be shown using a sequence of ZK greater than proofs on encrypted plaintexts for each two consecutive ciphertexts. This proof can be based on range proofs such as [Lip03].

A.2.2 Committed PRF Evaluations

The approach of running a protocol on committed inputs in order to enforce correctness is very well known in secure computation (see, e.g., [GMW87, JS07, DMV13]). In this section we deal with the case of the [NR97] PRF evaluation (see Section 2.4). The functionality for committed PRF evaluation is defined by,

$$\mathcal{F}_{\text{CPRF}} : ((k, x_1, \ldots, x_\tau), (\mathsf{Commit}(\mathsf{pk}, k), (\mathsf{Commit}(\mathsf{pk}, x_1), \ldots, \mathsf{Commit}(\mathsf{pk}, x_\tau)), (c'_1, \ldots, c'_\tau)) \mapsto (-, b)$$

where $b = 1$ only if $c'_i = \mathsf{Commit}(\mathsf{pk}, \mathsf{F}(k, x_i))$ for all $i \in [\tau]$. Namely, SEN commits to its PRF key, and inputs $x_1, \ldots, x_\tau$ for the PRF evaluations.

Recall that in the [NR97] PRF the key is a vector $k = (a_1, \ldots, a_m)$, and each input to the PRF is an m-length bit-string $x_i = (x_i[1], \ldots, x_i[m])$. The protocol we describe is for a setting where the verifier knows commitments c_j to each element in k, and commitments $c_{i,j}$ for each bit $x_i[j]$ of the inputs. (This is exactly the situation in the protocol of Section 5.3.) At first, the prover computes $e_{i,j} = a_j^{x_i[j]}$ for all $i \in [\tau]$, and the corresponding commitment $c^*_{i,j} = \mathsf{Commit}(\mathsf{pk}, e_{i,j})$. Next, the prover proves correctness of this computation to the verifier.[8] It then splits the values $\{e_{i,j}\}$ into distinct pairs, multiplies each pair, commits to the result, and proves correctness of this computation running protocol π_{mult} with the verifier. This step is run recursively until the exponent $\prod_j e_{i,j}$ is obtained, such that the correctness of the last step can be verified by inputting the value c'_i in protocol π_{mult}. Note that the above requires the invocation of the basic ZK proof π_{mult} for $O(\tau)$ times where the number of iterations is $O(\log \tau)$ for each i (we remark that the proofs can be invoked in parallel for all i since these are independent PRF evaluations).

A.2.3 Double Oblivious PRF Evaluation

Finally, in this section we discuss the implementation of the following functionality,

$$\mathcal{F}_{\text{2PRF}} : \big((k_0, k_1), (p, \mathsf{Commit}(\mathsf{pk}, k_0), \mathsf{Commit}(\mathsf{pk}, k_1))\big) \mapsto \big(-, \mathsf{F}(k_0, p), \mathsf{F}(k_1, p)\big)$$

which is an extension of the oblivious PRF evaluation functionality for which the receiver learns two PRF evaluations on a single element p (with two different keys k_0, k_1). The protocol from the previous section is not applicable here since the prover does not know p and thus cannot compute the PRF evaluation by itself. Instead, we extend the oblivious PRF solution of [FIPR05, HL10] to the double case. Namely, we invoke the basic protocol twice (where in each invocation the prover enters a different PRF key). In addition to that, we need to ensure that the prover indeed uses the correct PRF keys that were committed to within the commitments that the verifier holds. Briefly, for the above particular implementation of the [NR97] PRF this can be done by proving consistency between the ciphertext encrypting the PRF key within the oblivious PRF evaluation protocol, and the above commitments given in the state, using a ZK proof.

[8] The latter can be done by proving in ZK that a commitment to $e_{i,j} + x_i[j]$ is either a commitment to 1 or to $a_j + 1$. This implies correctness since the verifier knows the commitments of the bits in each x_i, thus the sum $a_j + 1$ can only equal 1 (in case $x_i[j] = 0$ and $e_{i,j} = a_j^0 = 1$) or $a_j + 1$ (in case $x_i[j] = 1$ and $e_{i,j} = a_j^1 = a_j$). This proof can be implemented efficiently with constant overhead if the commitment is homomorphic.